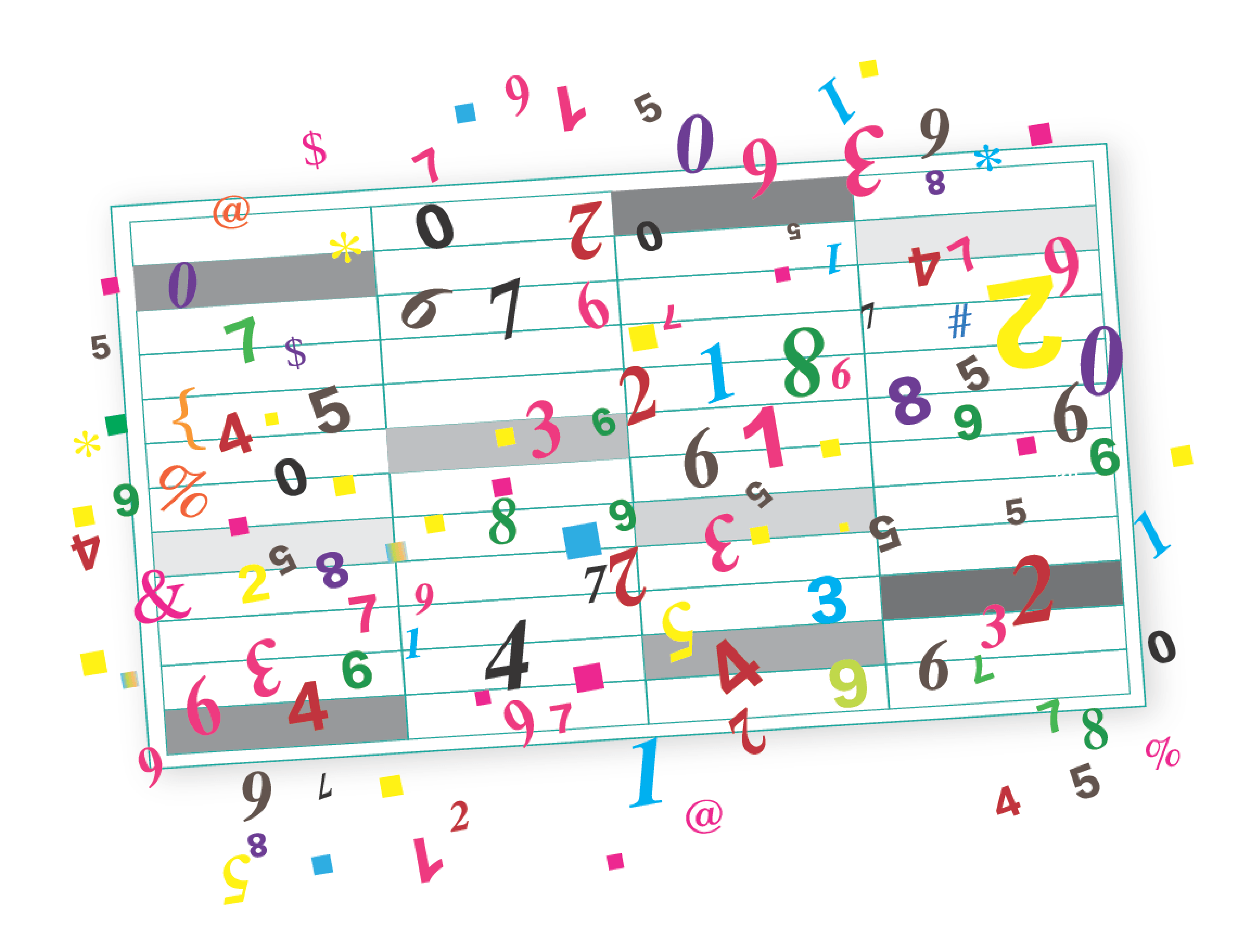


审计分析 从关系到大数据

董　东　王艳君　陈玉哲　编著

清华大学出版社

大数据系列丛书

审计分析：从关系到大数据

董　东　王艳君　陈玉哲　编著

清华大学出版社
北　京

内 容 简 介

本书针对计算机在审计中数据分析所需要的技术、方法和工具，按照技术发展的脉络，介绍基于关系数据库的以结构化查询语言(SQL)为工具的查询分析，基于数据仓库的可视化途径的多维分析，基于模型训练的机器学习途径的挖掘分析，以及基于大数据的相关分析，力图使读者能够应用本书介绍的方法、工具和技术完成审计目标。

全书共有7章。第1章介绍审计数据分析的基本概念；第2章以T-SQL为例介绍结构化查询语言，包括基本的DR、DDL、DML、DCL等，以及结构化查询语言在财务审计与业务数据结合的财务审计中的应用技术；第3章介绍数据库用户以及授权、数据导入导出等技术，以及这些技术在审计中的应用；第4章介绍高级查询分析技术，包括游标、触发器、视图、索引等；第5章介绍如何通过多维分析发现审计线索；第6章介绍数据挖掘途径的审计数据分析；第7章介绍基于大数据的审计分析。

本书可作为计算机审计工作者的技术参考书。

图书在版编目(CIP)数据

审计分析：从关系到大数据/董东，王艳君，陈玉哲编著.—北京：清华大学出版社，2019
(大数据系列丛书)
ISBN 978-7-302-52052-8

Ⅰ.①审…　Ⅱ.①董… ②王… ③陈…　Ⅲ.①审计学－研究　Ⅳ.①F239.0

中国版本图书馆CIP数据核字(2019)第009619号

责任编辑：张　玥　常建丽
封面设计：常雪影
责任校对：时翠兰
责任印制：李红英

出版发行：清华大学出版社
　网　　址：http://www.tup.com.cn，http://www.wqbook.com
　地　　址：北京清华大学学研大厦A座　　**邮　　编**：100084
　社 总 机：010-62770175　　**邮　　购**：010-62786544
　投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
　质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
　课件下载：http://www.tup.com.cn，010-62795954
印 装 者：北京鑫海金澳胶印有限公司
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张**：11.5　　**字　　数**：263千字
版　　次：2019年7月第1版　　**印　　次**：2019年7月第1次印刷
定　　价：49.00元

产品编号：079897-01

前　言

PREFACE

2000 年的春天，河北省审计厅陈金如副厅长来到河北师范大学，就计算机教育与培训问题与我进行了第 1 次“聊天”。当时没有想到，那次“聊天”却是十八年合作的开始。

当时，河北省审计系统面临如何提升全省审计工作者的计算机审计水平，以适应信息技术的迅猛发展问题。从事审计的人员如果不懂信息技术，那么在信息化环境下就进不了被审单位的门，打不开被审单位的账。审计署指出，开展计算机审计要靠审计人员素质的提高，靠审计机关知识结构的改善。根据审计署的要求，河北省审计厅认为计算机审计的发展和审计事业的发展已经密不可分，应从事关全省审计事业发展的高度思考计算机审计问题，强力推进计算机审计。应切实增强危机感和紧迫感，下大力量将计算机辅助审计工作搞上去。

2000—2005 年，我应邀为审计厅的几次短期培训授课。从 2006 年开始，河北省审计厅每年都与河北师范大学联合举办“河北省审计机关计算机审计中级培训”，目前仍然在进行。参训学员大部分已成为各单位审计信息化工作骨干，并多次在审计署组织的 AO 应用实例评选和计算机审计专家经验评选中获奖。

在培训项目中，我不仅负责组织工作，而且主讲了“数据库应用技术”“审计数据高级分析技术”两门课程。为了把审计业务与计算机技术结合起来，让非计算机专业的审计师能够理解、掌握和应用计算机技术，在省审计厅安排下，我还深入审计现场，实地调研，学习审计业务，了解审计人员的需求；钻研审计一线人员提出的技术问题，提出解决方案；不断研究数据仓库、数据挖掘、大数据等技术在河北省计算机审计业务中的应用等问题，并将研究结果应用于培训。2014 年，受河北省审计厅计算机中心委托，编写了“技巧系列丛书”中的《数据挖掘技巧》一书的初稿，并根据审计署、省审计厅的意见进行了修改，于 2017 年出版。十八年来累积的讲义以及其他技术资料经过整理，形成了本书。本书的内容不仅涉及与计算机审计有关的技术，还包括如何规避常见的技术错误。

本书初稿由河北师范大学的董东老师完成。河北师范大学的王艳君、陈玉哲、张朝昆等老师在历年培训过程中对前 4 章进行过修订。王艳君对本书的章节组织以及第 1～4 章内容提出了修改意见；陈玉哲对第 5～7 章提出了修改意见。董东最后对全稿进行了精简。

本书得到河北师范大学应用开发基金（L2013K01）和河北省审计厅 2018 年重点科研课题（201805）资助，在此表示感谢。

囿于学识，在思想、方法或者技术等方向定有不当之处，望读者批评指正。

董　东

2018 年 12 月于河北师范大学

目 录

CONTENTS



审计数据分析

数据分析在计算机科学领域和统计学领域已经被广泛关注并取得了大量研究成果，并在各种领域已经得到广泛应用以解决本领域的具体问题，如商品推荐、交通管理、舆情分析等。

审计是对业务实体经济活动的真实性、合法性和效益情况的验证过程。计算机审计是以计算机为基本工具，以业务系统、财务收支系统电子数据为主要对象，通过计算机软件工具的功能与审计人员经验判断的结合，实现对业务实体经济活动进行真实、合法、效益的信息化审计过程。在“金审工程”推动下，我国计算机审计的研究和应用发展较快。但是，如何应用数据分析技术有效地实施计算机审计仍然是一个热点问题。

审计数据分析活动主要是验证(validation)活动，即按照政策、法规对反映经济活动数据项或者数据处理进行检查，以查证是否为完全按照政策、法规执行的活动，即查证被审单位是否正确进行了经济活动。

随着信息技术的普及和广泛应用，企事业单位每天产生的经济业务活动均能够以不同形式被各种各样的信息系统记录下来。那么，从这些纷繁复杂而且大量的数据中完成审计任务和达到审计目标成为当前审计领域关注的重要问题之一。本书讨论从数据中发现审计线索的相关数据分析概念、技术和方法。

1.1 审　　计

国务院设立审计机关，对国务院各部门和地方各级政府的财政收支，对国家的财政金融机构和企业事业组织的财务收支，进行审计监督。审计机关通过客观地获取和评价有关经济活动与经济事项认定的证据，以证实这些认定与既定标准的符合程度，并将结果传达给有关使用者。

审计的主体是从事审计工作的专职机构或专职的人员，是独立的第三者，如国家审计机关、会计师事务所及其人员。审计的客体是被审计单位的财政、财务收支及其他经济活动。审计的基本工作方式是搜集证据，查明事实，对照标准，做出好坏优劣的判断。审计的主要目标不仅要审查评价会计资料及其反映的财政、财务收支的真实性和合法性，而且还要审查评价有关经济活动的效益性。

1.2 计算机审计

计算机审计是以计算机为基本工具，以被审单位的财务收支电子账和业务系统的数据库为主要对象，在计算机软件工具的辅助下结合审计人员经验，对业务实体经济活动进行的真实、合法、效益的信息化审计过程。计算机审计丰富了传统审计中各阶段的内容。一个计算机审计项目主要包括 3 个阶段：准备阶段、实施阶段和报告阶段。

在审计准备阶段，审计人员和信息技术人员需要到被审计单位了解信息系统、业务过程、电子数据等状况；调查系统结构、业务处理流程、数据存储结构及数据处理流程；从被审单位系统采集审计相关的会计核算数据和业务数据；将数据导入审计软件；验证数据的合法性、有效性与完整性。

在审计实施阶段，根据审计人员经验和相关技术方法设计模型，对审计数据进行查询分析、多维分析或者发掘分析，发现审计线索。然后寻找疑点、落实线索形成证据。

在审计报告阶段，形成审计报告，送达有关使用者。还将被审单位系统、数据情况，以及操作流程加以归档，供以后审计使用。具体包括：被审单位信息系统开发及应用情况；业务流程及数据流程文档；与审计数据相关的数据结构资料；本次审计所建立、应用的审计模型，发现问题的方法思路及编写的程序或操作步骤。

1.3 审计数据分析

根据技术和模型的不同，审计数据分析分为 4 类：基于数据库的审计分析、基于数据仓库的审计分析、基于数据挖掘的审计分析和基于大数据的审计分析。

基于数据库的审计分析也称为查询型分析，指建立模型、设计结构化查询语言(Structured Query Language，SQL)语句查询数据库、分析查询结果的过程。查询型分析的主要对象是关系数据库管理系统中的表，涉及单表查询、多表查询等技术。一个成功的查询分析既需要定义一个很好的业务问题(查什么)，也需要设计很好的 SQL 语句(如何查)。

当需要从不同角度可视化地观察某个时期、某个主题的历史数据时，就需要应用基于数据仓库的分析技术。多维分析模型描述了如何在多个维上观察业务事实。事实是一组与业务事务或者事件相关的数据项。例如，会计事务中的一笔凭证就是一个事实，这个事实中含有日期、凭证类型、部门、科目、借贷金额等数据项。维是一组对事实进行分析所使用的属性，如在日期和科目属性上对凭证事实进行分析。

很多情况下，审计活动是基于经验的。经验是审计人员在长期实践中经过归纳、推演发现的一些特征或者规律。基于数据挖掘的分析指先由计算机发现隐藏在数据中的、不为审计人员预先所知的规则、规律、模式或者趋势，然后审计人员对这些发现的计算机“经验”进行甄别和利用。

大数据指数据量大得超出了常规软件的管理能力的数据。大数据分析的策略不同于常规数据。这些策略有基于总体而不是基于抽样，基于相关而不是基于因果，等等。

基于关系数据库的审计分析、基于数据仓库的审计分析、基于数据挖掘的审计分析和基于大数据的审计分析一起构成了审计数据分析，但它们有各自的适用范围。基于关系数据库的审计分析是已知数据结构的情况下，对操作型数据的访问，比较容易建立模型；基于数据仓库的审计分析需要对被审数据重新组织，需要从不同角度观察数据，建立模型较为复杂；而基于数据挖掘的审计分析需要审计人员根据具体业务问题选取合适的挖掘模型，从而在数据挖掘工具帮助下发现有用模式和趋势，处于高级的分析层次；基于大数据的审计分析的关键是获取数据的总体，有了总体，就可以用简单的模型解决复杂的问题。

1.3.1 基于关系数据库的审计分析

从集合角度看，关系(relation)是元组(tuple)的集合。一个元组中有若干属性(attribute)值。在关系中能唯一标识一个元组的属性集称为关系的超键(super key)。不含多余属性的超键称为候选键(candidate key)。一个关系中可能存在多个候选键，用户选作元组标识的候选键称为主键(primary key)。

例如，有关系：医生(医生号，姓名，性别，年龄，职称，身份证号，部门号)，其中“医生”是关系的名称；医生号、姓名、性别、年龄、职称、身份证号、部门号是该关系的属性。这些属性中，医生号和姓名两个属性一起可以作为超键，姓名和身份证号这两个属性一起也是超键。而医生号是候选键，身份证号也是候选键。可以把“医生号”作为主键，也可以把“身份证号”作为主键，但一个关系上只能有一个主键。

关系的属性值不可分解，从而不允许表中有表；一个关系中的元组不可重复；关系是元组的集合，集合的元素是无序的；属性之间也无序。

在关系上可以施加3类完整性约束：实体完整性(entity integrity)、参照完整性(referential integrity)和用户定义的完整性。

1. 实体完整性

若属性A是关系R主键中的属性，则属性A不能取空值。一个元组对应现实世界的一个实体，一个关系对应现实世界的一个实体集。现实世界中的实体是完整的，即每个实体都具有属性值。而在关系中不可能穷尽所有的属性，只能有感兴趣的部分属性。无法要求这部分属性都必须有属性值，但可以要求主键必须有值，即不能取空值(NULL)。所谓空值，就是“不知道”或“无意义”的值。如果主键中的属性取空值，就说明存在某个不可标识的实体，即存在不可区分的实体。例如，医生(医生号，姓名，年龄，身份证号)中，如果医生号为主键，则该属性不能取空值。

2 参照完整性

设F是关系R的一个或一组属性，但不是关系R的键，如果F与关系S的主键对应，则称F是关系R的外键(foreign key)。例如，医生(医生号，姓名，部门号)和部门(编号，名称，级别)两个关系，“部门号”是医生关系模式的外键，对应于“部门”关系的属性“编号”。

若属性(或属性组)F是关系R的外键，它与关系S的主键对应(关系R和S可能是

同一个关系），则对于 R 中每个元组在 F 上的值都必须或者取空值（F 的每个属性值均为空值），或者等于 S 中某个元组的主键值。

例如，对于“医生”关系中的每个元组，其在“部门号”上的取值要么是空值，要么是“部门”关系中某一元组在“编号”属性上的值。也就是说，某个医生所在的部门必须是存在的一个部门。

在“医生”和“部门”这两个关系中，为了维护参照完整性，删除“部门”关系中的元组时可以采用 3 种策略：级联删除、级联受限删除和置空值删除。例如，删除“部门”关系中的 10101 部门，级联删除是将关系“医生”中所有部门号（外键）为 10101 的元组删除，接着将关系“部门”中键为 10101 的元组也删除；受限删除是当关系“医生”中没有任何元组的部门号（外键）为 10101 时，才执行删除操作，否则拒绝删除；置空值删除则将关系“医生”中部门号（外键）为 10101 的元组的部门号置空，然后删除关系“部门”中键为 10101 的元组。

如果要在关系“医生”中插入元组，其“部门号”在“部门”关系中存在，则插入操作可以顺利执行；如果不存在相应的元组，则可以有两种策略：受限插入和递归插入。前者拒绝在“医生”关系中插入元组；后者首先向“部门”关系中插入相应的元组，其编号（主键）值等于“医生”关系插入元组的部门号（外键）值，然后向参照关系中插入元组。

3. 用户定义的完整性

用户定义的完整性就是针对某一具体关系数据库的约束条件，它反映某一具体应用涉及的数据必须满足的语义要求。关系数据库的实现提供定义和检验这类完整性的机制，以便用统一的、系统的方法处理它们，而不要由应用程序承担这一功能。例如，约束“年龄”属性是 0～200 的一个整数。

数据库（database）是数据的汇集，它以某种组织形式存储在介质上。数据库管理系统（Database Management System，DBMS）是管理数据库的系统软件。从操作系统角度看，数据库表现为一个或一组特定扩展名的文件。例如，一个 Access 数据库是一个扩展名为.mdb 的文件，再如，一个 SQL Server 2008 的数据库在最简单的情况下由两个文件构成，一个文件的扩展名为.mdf，另一个文件的扩展名为.ldf。

按照关系模型实现的数据库关系系统称为关系数据库管理系统。目前广泛使用的数据库管理系统有甲骨文公司的 Oracle、Microsoft 公司的 SQL Server、IBM 公司开发的 DB2 以及开源数据库 MySQL 等。由于大多数的数据库管理系统产品都是关系模型的实现，所以后文中的“数据库管理系统”默认指“关系数据库管理系统”。

图 1-1 展示了关系模型中的术语与关系数据库中的术语对应关系。

从图 1-1 中可以看到，一个关系对应数据库中的一个表；若干关系组成关系模式，若干表形成数据库模式；一个元组对应表中的一行，元组在某个属性上的值对应行在某个列上的值。

数据库管理系统的主要功能有：定义数据库、操纵数据、控制数据库和维护数据库。

1）定义数据库

DBMS 提供数据描述语言（Data Definition Language，DDL），定义数据的结构、数据与数据间的关系、数据的完整性约束等。数据库中的主要对象是表（table），数据组织在

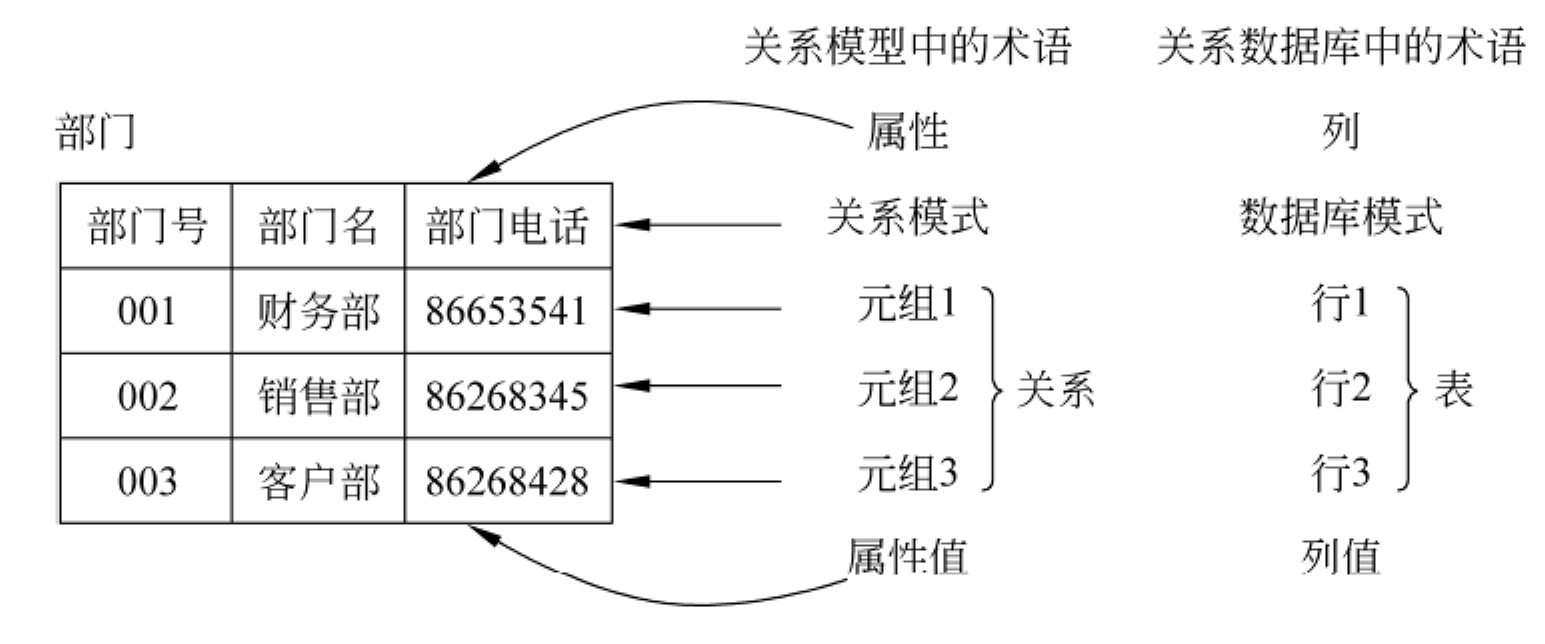


图 1-1 关系模型中的术语与关系数据库中的术语对应关系

表中。数据的完整性约束既可以定义在表上,也可以定义在表之间。数据库中的表、约束等对象合起来称为数据库模式(schema)。

2) 操纵数据

DBMS 提供数据操纵语言(Data Manipulation Language,DML),可实现对数据的查询、插入、删除和修改等操作。DML 有两种用法:一种方法是把 DML 语句嵌入到高级语言中;另一种方法是交互式地使用 DML 语句。对于第一种方法,DBMS 必须提供预编译程序,预处理嵌入 DML 语句的源程序,识别 DML 语句,转换为相应高级语言能调用的语句,以便原来的编译程序能接受和处理它们。

3) 控制数据库

数据库的控制功能包括并发控制、数据的安全性控制、数据的完整性控制和权限控制,保证数据库系统正确有效地运行。

4) 维护数据库

已经建立好的数据库,在运行过程中需要进行维护。维护功能包括数据库出现故障后的恢复、数据库的重构、性能的监视等。这些功能大部分由实用程序完成。

数据字典(Data Dictionary,DD)中存放着数据库体系结构的描述。对于用户的一个处理请求,DBMS 都要查阅数据字典。

当应用程序需要处理数据库中的数据时,首先向数据库管理系统发送一个数据处理请求,数据库管理系统接收到这一请求后,对其进行分析,然后执行数据操作,并把操作结果返回给应用程序。由于应用程序直接与用户打交道,而数据库管理系统不直接与用户打交道,所以前者常被称为“前台”,后者常被称为“后台”。

由于应用程序是向数据库管理系统提出服务请求,通常称为客户机(Client)程序,而数据库管理系统是为其他应用程序提供服务,通常称为服务器(Server)程序,所以又将这种实现模式称为客户机/服务器(C/S)模式。

对于一个应用系统,需要有哪些表?表之间是什么关系?即如何设计数据库模式?数据库模式一般从概念模型得到。

概念模型用于信息世界的建模。概念模型不依赖于具体的计算机系统。概念模型可以转换为计算机上某一 DBMS 支持的特定数据模型。

在概念模型中,客观存在并可相互区别的事物称为实体(entity)。实体可以是具体的

人、事、物，也可以是抽象的概念或联系。例如，医院中的一名医生、一个部门都是实体。实体具有的某一特性称为属性(attribute)。一个实体可以由若干个属性刻画。例如，要在数据库系统中记录一名医院的医生，可以包括一组属性：医生号、姓名、性别、年龄、职称、身份证号等；要记录一个凭证，可以用凭证号、借方发生额、贷方发生额等属性。属性的取值范围称为该属性的域(domain)。

唯一标识一个实体的属性集称为键(key)。键是一个属性集，属性集可能由单个属性构成，也可能由多个属性构成。例如，医院中，“医生号”可以作为“医生”的键，这时键由单个属性构成；要标识一张火车票，则需要日期、车次、车厢、座位号，这时键由一组属性构成。键的确定是一个语义范畴的问题。实体可能同时存在多个键。例如，“医生号”可以作为医生的键，而“身份证号”也可以作为医生的键。当实体有多个键时，选其中一个键作为主键。

用实体名及其属性名集合抽象和刻画同类实体，称为实体型(entity type)。例如，医生的实体型可以表示为：医生(医生号，姓名，性别，年龄，职称，身份证号)，属性之间用逗号隔开。同型实体构成的集合称为实体集(entity set)。

现实世界中，事物内部以及事物之间的联系在信息世界中反映为实体内部的联系和实体之间的联系(relationship)。两个实体集之间的联系有 3 种类型：一对一联系、一对多联系和多对多联系。

如果对于实体集 A 中的每一个实体，实体集 B 中至多有一个实体与之联系，反之亦然，则称实体集 A 与实体集 B 具有一对一联系，记为 1∶1。例如，一份审计报告只涉及一个审计项目，而一个审计项目最终产生一份审计报告。

如果对于实体集 A 中的每一个实体，实体集 B 中有 n 个实体($n\geqslant 0$)与之联系，反之，对于实体集 B 中的每一个实体，实体集 A 中至多只有一个实体与之联系，则称实体集 A 与实体集 B 有一对多联系，记为 $1:n$。例如，处室和工作人员的联系。一个处室中有若干工作人员，一个工作人员至多在一个处室中工作，由处室到工作人员是一对多联系。需要注意的是，一对多联系不是对称的，由处室到工作人员是一对多联系，反过来，由工作人员到处室是多对一联系。

如果对于实体集 A 中的每一个实体，实体集 B 中有 n 个实体($n\geqslant 0$)与之联系，反之，对于实体集 B 中的每一个实体，实体集 A 中也有 m 个实体($m\geqslant 0$)与之联系，则称实体集 A 与实体集 B 具有多对多联系。记为 $m:n$。例如，审计小组和工作人员的联系。一个审计小组由多个工作人员组成；一个工作人员可以参加多个审计小组。再如，一个审计小组可以负责多个审计项目；一个审计项目可以由多个审计小组负责。

实体-关系图(E-R 图)提供了表示实体型、属性和联系的可视化方法。在 E-R 图中，使用矩形表示实体型，矩形框内写明实体名。例如，表示医院的医生和部门：

医生	部门

用椭圆表示属性，并用无向边将其与相应的实体连接起来。例如，医生和部门的属性表示如图 1-2 所示。

使用菱形表示联系，菱形框内写明联系名，并用无向边分别与有关实体集连接起来，

同时在无向边上标记联系的类型($1:1$、$1:n$ 或 $m:n$)。例如,医生和部门的联系如图 1-3 所示。

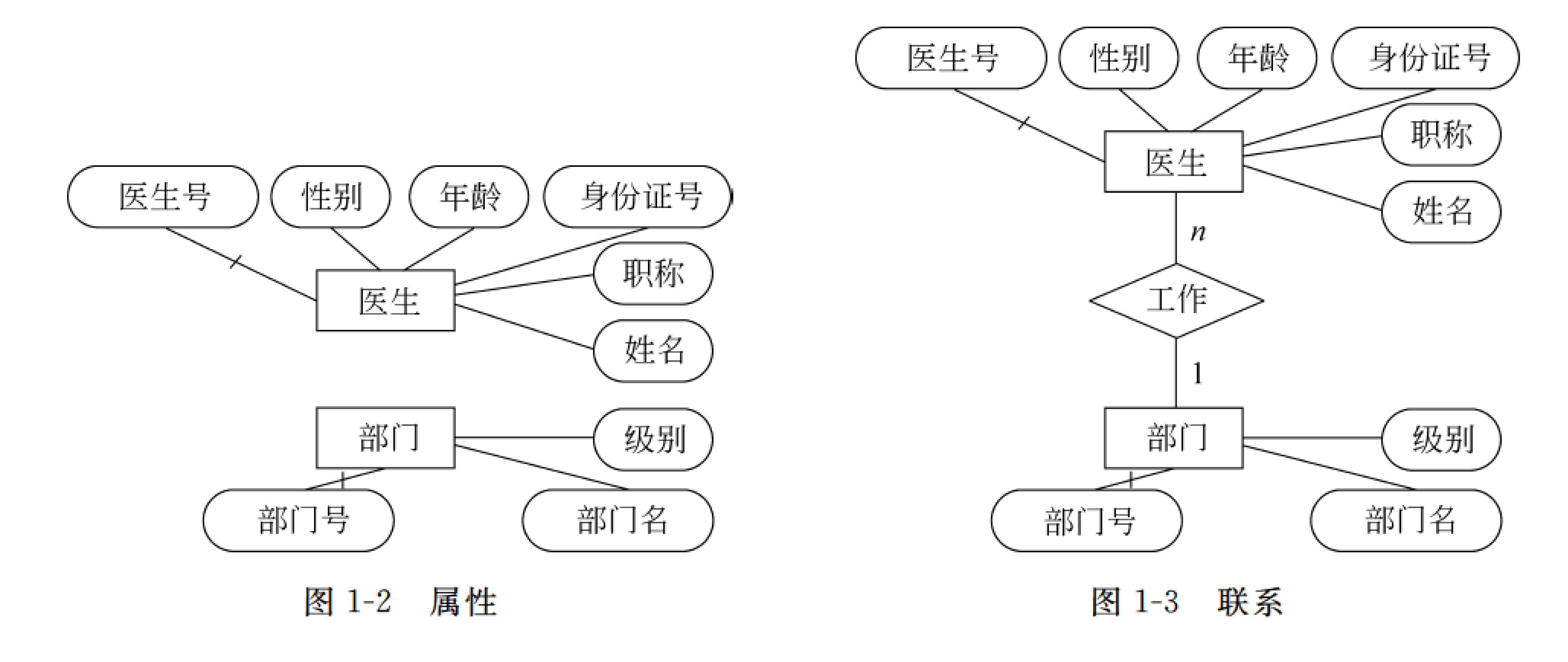


图 1-2　属性　　　　图 1-3　联系

联系本身也是一种实体型,也可以有属性。如果一个联系具有属性,则这些属性也要用椭圆形表示,并用无向边与该联系连接起来。

联系存在于两个实体之间,也可以存在于 3 个实体之间。例如,一次住院收费涉及 3 个实体:医生、患者和收费项目,可以用图 1-4 描述这三者之间的联系。

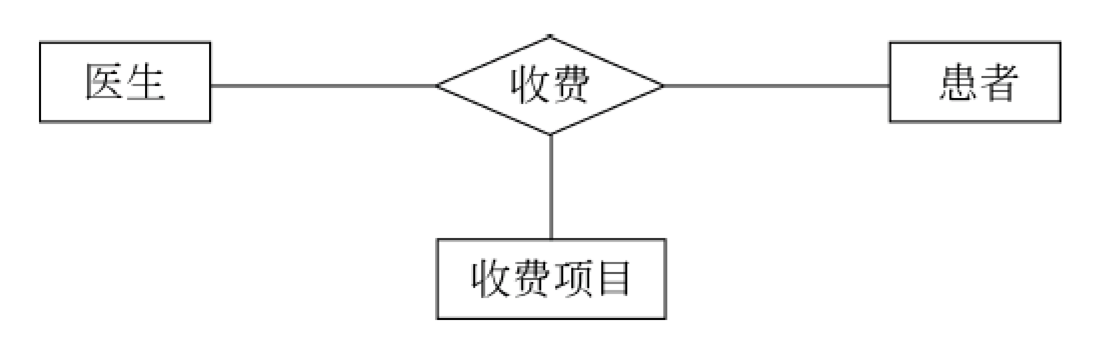


图 1-4　住院收费之间的联系

图 1-5 展示了两个实体集之间每种联系的 E-R 图表示方法。

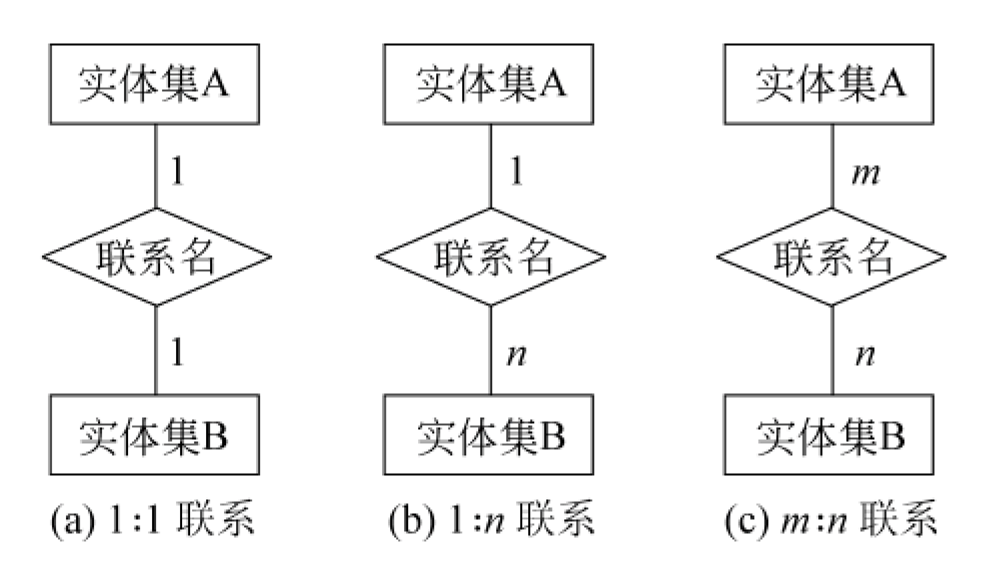


图 1-5　3 种联系的表示

通常将实体集及实体集间联系的上述表示模型称为实体-关系(relationship)模型;从分析用户项目涉及的对象及对象之间的联系出发,到获取 E-R 图的这一过程称为概念模型设计。

查询型分析指通过结构化查询语言(Structured Query Language,SQL)形成 SQL 语句交互式查询数据库的过程。查询型分析的主要对象是数据库管理系统中的表。运用 SQL 语句进行有关数据采集、数据预处理和数据分析在我国面向数据的计算机辅助审计

中的应用十分广泛，出现了大量成功的案例。数据查询语句，包括单表查询、面向多表的连接查询和嵌套查询是实现查询分析的主要技术。其他技术还有：使用行选择条件、分组、分组选择条件、排序、选择前若干行查询结果等；使用统计函数进行全表统计和分组统计；将多个查询语句的结果合并为一个结果；如何将查询结果保存在永久表和临时表中等。另外还有使用插入语句、数据更新语句和数据删除语句进行数据操纵。

查询型分析的技术要点是 SQL。而应用 SQL 解决实际问题的前提是理解关系模型和关系数据库模式。

1.3.2 基于数据仓库的审计分析

数据仓库就是面向主题的、集成的、相对稳定的数据集合。主题是关注的问题，一个主题对应一个宏观的分析领域。数据仓库中的数据是从原有分散的面向联机事务处理的数据库中抽取出来的。由于数据仓库的每一主题对应的源数据在原有分散的数据库中可能有重复或不一致的地方，因此数据在进入数据仓库之前必须经过数据的抽取、清洗和转换。数据仓库中存放的是历史数据，而不是日常事务处理产生的数据，数据进入数据仓库后极少甚至根本不被修改。

多维数据模型提供了多角度、多层次的分析应用，如基于时间维、地域维等构建星形模型或者雪花模型，可以实现在各时间维度和地域维度的交叉查询。多维分析技术以及分类技术、聚类技术等数据挖掘技术已经开始应用于审计分析中。多维分析途径的计算机审计过程如图 1-6 所示。

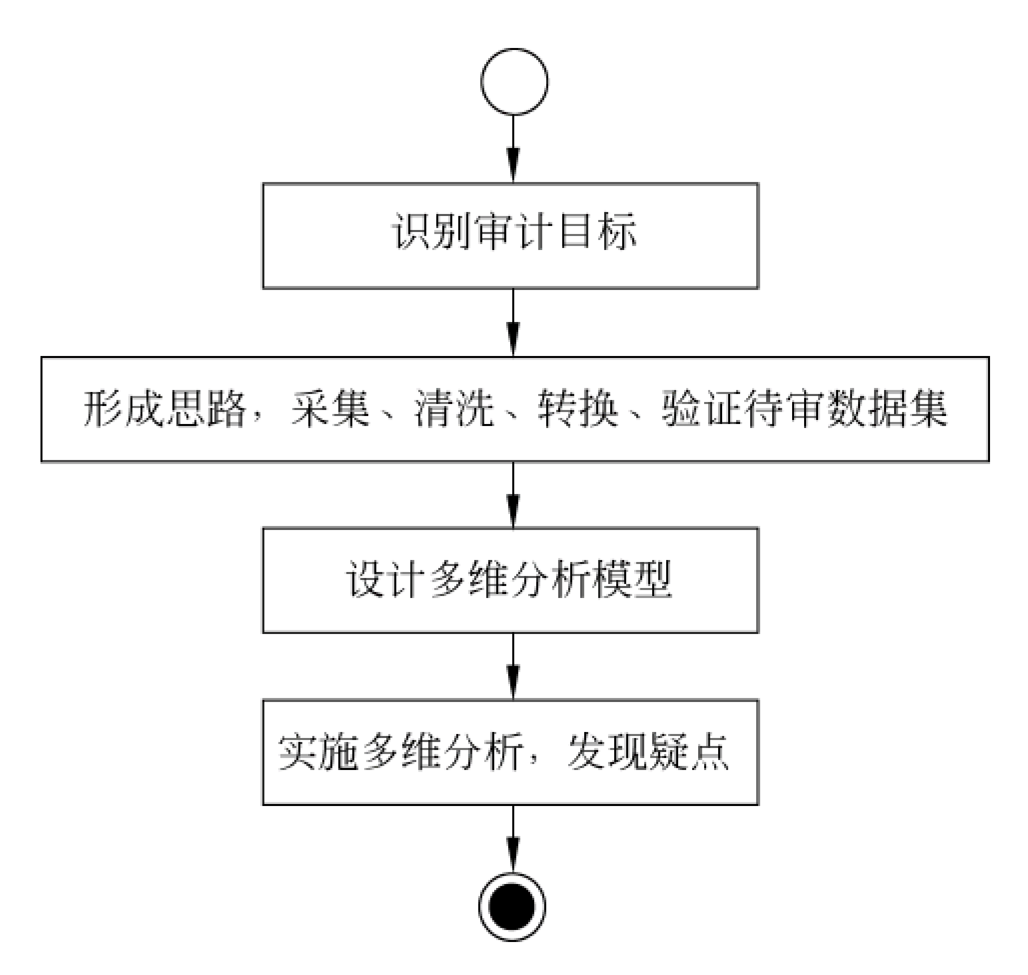


图 1-6　多维分析途径的计算机审计过程

例如，税务机关可以批准企业延期纳税，一个可能的审计目标是查证有无违法批准延期纳税的疑点。于是审计人员设想，能否从若干税种中选定一个常见的税种，如“企业所得税”，然后从若干税务机关中选择一个或者若干个，观察这些税务机关在某年某月某日的纳税额，从企业缴纳的时间、数额特征中寻找线索，这就是审计思路。

有了思路，就按照这个思路从税务机关采集、清洗和转换数据，存储到数据库或者数

据仓库中，整理成事实表。然后设计多维分析模型。想法中的“税种”“税务机关”和“日期”就是多维分析模型中的 3 个“维”；而纳税额就是多维分析模型中的“度量值”。

建立好多维分析模型后，应用软件工具以多维方式浏览数据，观察可疑模式，发现审计疑点。

1.3.3 基于数据挖掘的审计分析

从审计角度看，数据挖掘就是根据事先明确的审计目标，对被审单位的大量业务数据进行分析，揭示其中潜在的逻辑关系和规律，进而形成明确而有效的审计思路的过程。例如，银行贷款五级分类真实性情况审计。由于种种原因，某些事实上的不良贷款（次级、可疑或损失）被商业银行人为划归正常贷款分类（正常或关注）中。按照传统的审计思路，需要与被审单位了解贷款五级分类规则，然后根据被审单位提供的规则逐一核实各笔贷款的分类是否正确。挖掘型分析途径是先让计算机从被审单位提供的大量无错误分类的贷款数据中学习到贷款五级分类规则，然后应用该规则到被审贷款数据集上，从而发现审计线索。

数据挖掘途径的计算机审计过程如图 1-7 所示。

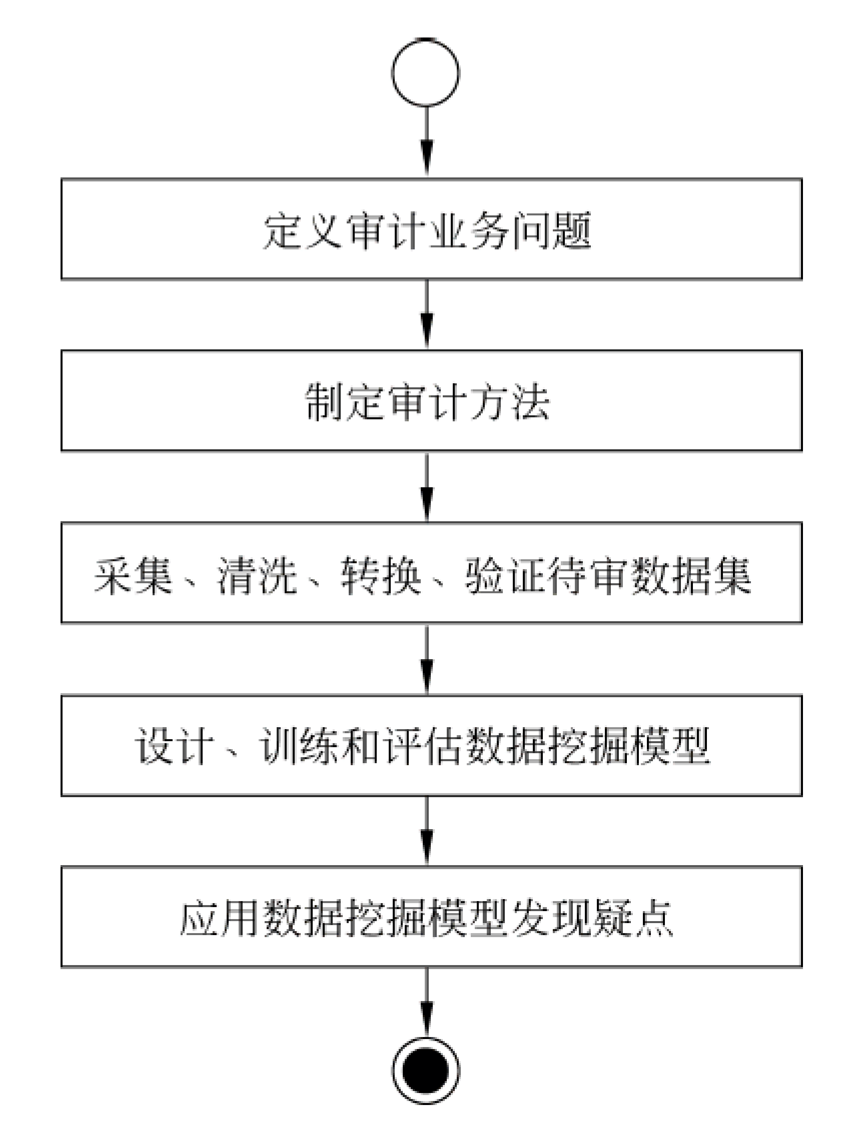


图 1-7 数据挖掘途径的计算机审计过程

“对某省某商业银行商业贷款进行五级分类真实性核实”就是一个审计目标。为了完成这个目标，首先采集、清洗和转换商业贷款以及五级分类情况的数据。因为这是数据挖掘中的“分类”问题，所以选择某种分类学习算法，如决策树算法，在具有分类标签的无错误分类的数据集上学习分类规则，并对学习到的分类规则进行评估。如果评估认为机器学习到的分类规则与实际的分类规则具有较高的一致性，则应用分类规则到可能具有错误分类的商业贷款事例上，即应用规则重新对这些商业贷款事例进行分类。重新分类得到类标签与原来的类标签不一致的事例就形成了审计疑点。

1.3.4 基于大数据的审计分析

审计数据获取、数据可视化技术、审计人员能力提升是当前审计行业共同关注的问题。如今，大数据环境为审计工作提供了总体数据，使得审计全覆盖成为可能。

大数据背景下，审计工作的展开是数据导向的，将会形成自顶向下、逐渐细化的工作模式：总体把握→发现疑点→分散核实。在大量数据中进行可视化分析，以把握总体、数据挖掘分析，以发现要点，从大量数据中发现审计重点、疑点，更加有针对性地开展现场审计，提升审计效率。

当数据集的规模超出了传统数据库软件的管理和分析能力，通常达到几十 TB，甚至

几个 PB 规模时，被称为“大数据”。大数据有 3 个基本特征：体量（volume）大、流动（velocity）快和多样性（variety）。随着物联网技术、互联网技术不断深入人类社会的各个角落，人类的社会活动，无论是步行过马路、还是驾车过道口；无论是使用社交软件聊天，还是网上浏览购物；无论是使用网约车出行，还是到外地住宿餐饮，这些行为都被以各种各样形式记录下来归结到一起，就形成了大数据。

大数据为审计提供了全覆盖的机遇，也提出了新的挑战：如何从大体量的、纷繁复杂的经济活动中发现审计疑点和审计线索。例如，如何从海量贷款示例中查找出异常贷款？

“自顶向下、逐层细化”是基于大数据审计分析的一种策略，即先把握总体；然后发现疑点，即找到突破口；最后定位事项，分散核实，归结证据。

把握整体就是借助可视化的数据分析工具选取关键的业务、财务特征，以二维或多维形式对审计数据进行多维度可视化分析，观测被审计数据在时间、空间上的分布以及变化趋势；比较同比、环比的变化，发现大的波动，确定主题，在整体和宏观层面上通过观察发现规律，寻找异常。

把不同部门的不同主题的数据关联起来、使用适当的特征，利用结构化查询技术或者数据挖掘技术发现疑点。例如，城乡低保资金管理使用合规情况审计来说，从不动产登记、公安户籍管理等多个部门采集数据，与低保数据进行关联，可以查出死亡冒领低保金、不符合条件领取低保金、重复享受城乡低保待遇等问题。

最后，根据疑点逐一分散调查取证，进行核实。

1.3.5 审计方法模型

无论采用何种技术进行计算机审计，在定义目标之后都要确定审计思路，制定审计方法。数据流程图是对某一计算机审计思路的图形表示，图中的符号包括三类。

(1) 表示数据存储的数据符号。

(2) 表示对数据执行某种操作的处理符号。

(3) 表示处理步骤或数据间数据流向的流线符号。

图 1-8 是一个数据流程图样例。虚线框中的文字是对流程图中各个符号含义的说明。

流线指示数据流或控制流。流线可以有箭头，表示数据流或者控制流的方向。控制流指示先做什么，后做什么。更为具体的符号参见《计算机审计方法流程图编制规范——计算机审计实务公告第 12 号》。

绘制数据流程图时，应注意以下几点：

(1) 在处理符号的前后都应是数据符号。

(2) 在图中应对各个符号均匀地分配空间，连线应保持合理长度，尽量少使用长线。保持图在整体上左右对称。

(3) 使用各种符号时必须按照规范所给符号的形状，尤其不要改变角度和其他影响符号形状的因素，统一各种符号的大小、粗细等样式。

(4) 把理解某个符号的功能所需要的最低限度的说明性文字置于符号内。文字应该按从左至右和自上向下的方式书写，与流向无关。若说明性文字的篇幅很大而不便放进

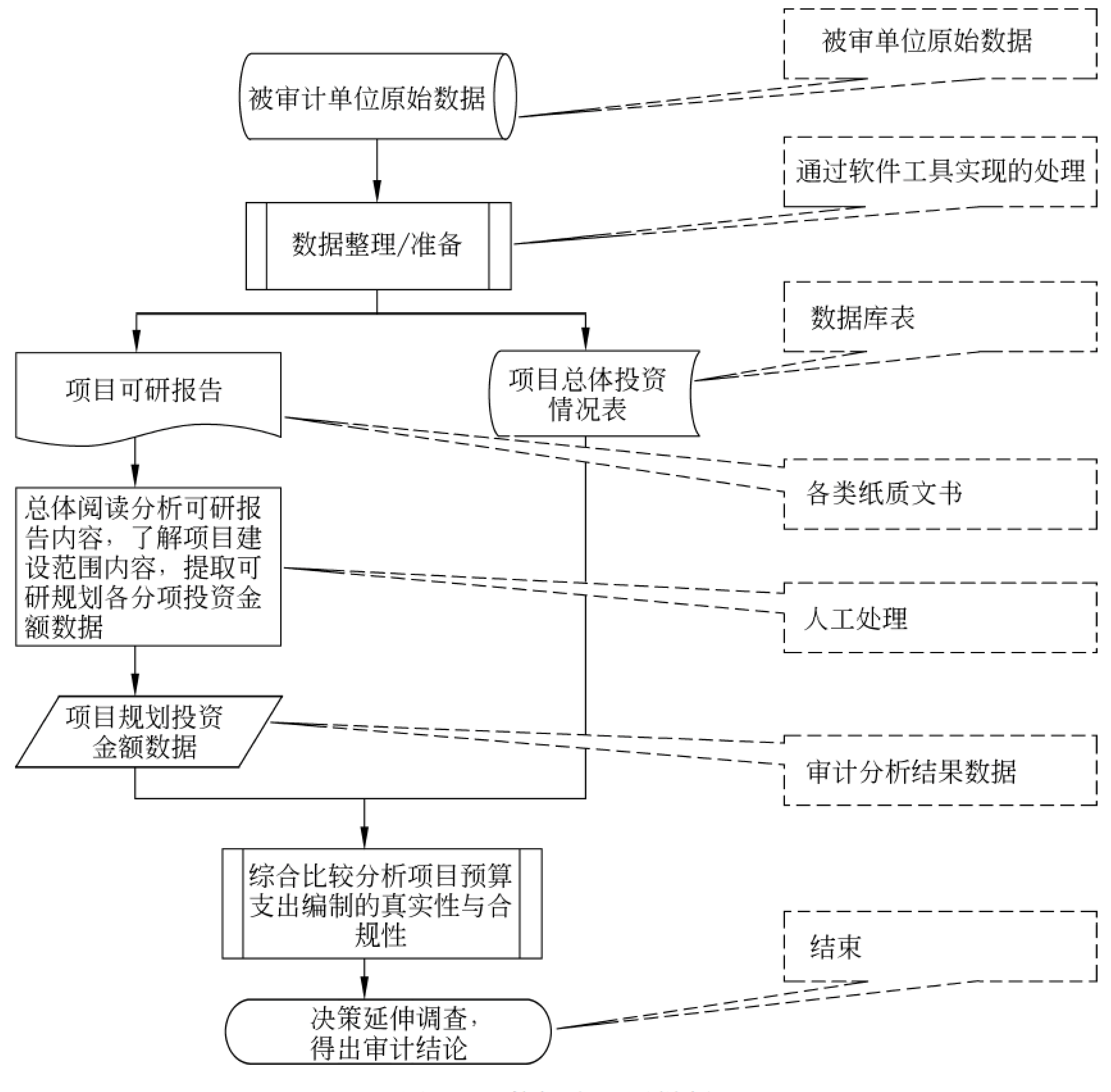


图 1-8　数据流程图样例

符号时，可以使用一个注解符号。

（5）控制流或者数据流的流向要么从左到右，要么自上而下。使用箭头指示流向。

（6）尽量避免流线交叉。即使出现流线的交叉，交叉的流线之间也没有任何逻辑关系，不对流向产生任何影响。

第2章

结构化查询技术及其应用

结构化查询技术是实现基于关系数据库的审计分析基本技术。审计师可以在客户端工具中直接与数据库服务器连接发送查询语句，观察查询结果；也可以在 AO(审计师办公室)软件系统中把查询语句嵌入在由 ALS 语言编写的审计脚本中，实现在客户端工具中嵌入式数据查询。

结构化查询的主要操作对象是关系数据库中的表，主要操作是对表中的行进行选择(SELECT)。本章以查询“东山县医院财务库”和“东山县医院业务库”两个数据库为例，介绍基本的和常用的查询技术。

2.1 概　　述

在关系数据库中，信息被组织成若干表的集合。表是行的集合。每一行中包含了若干列值。例如，表 2-1 共有 10 行，每行由“药品编号”“名称”“单位”和“单价”4 个值。

表 2-1　药品

药品编号	名　　称	单　位	单价/元
A010001	青霉素(甲)	支	0.6300
A010002	氨苄青霉素(甲)	支	0.8800
A010003	青坦威(乙)(凯德林)	支	38.0000
A010004	链霉素针(甲)	支	0.9000
A010005	洁霉素针(甲)(林可霉素)	支	0.6750
A010006	氯霉素针(甲)	支	0.3600
A010007	红霉素针 0.25g	支	1.2600
A010008	红霉素针(甲)30 万	支	4.4300
A010009	丁胺卡那针(甲)(阿米卡星)	支	1.3300
A010010	庆大霉素针(甲)	支	0.2700

为了从表中检索特定的行，如庆大霉素针(甲)，用户必须告诉数据库管理系统去做什么。SQL 就是数据库用户命令数据库管理系统实现其意图的语言。例如，检索庆大霉素

针(甲)的价格及其他所有信息的命令是

```
SELECT *
FROM 药品
WHERE 名称='庆大霉素针(甲)'
```

SQL 是一个通用的功能极强的关系数据库标准语言。使用 SQL 不需要表达如何访问数据库,只表达需要数据库管理系统做什么即可。

在表 2-1 所示的例子中,表中只有 10 行,其实不用 SQL 语句,通过审计人员的眼睛人工搜索,也能很快找到"庆大霉素针(甲)"。对于只有 10 行的表,甚至对于只有 100 行的表,人工搜索确实可以。但是,当面对一个 10000 行的表,甚至一个 1000000 行的表,采用人工搜索的办法就不合适了,必须依靠 SQL 这样的工具实现快速检索。

通过 SQL 操纵的基本对象是表,表是数据库管理系统中的概念。从用户角度看,数据库中包含了若干表;从操作系统角度看,数据库是操作系统中的一个或一组磁盘文件。与普通磁盘文件不同的是,这些文件的大小是预先分配的,即用户创建数据库时就把磁盘空间分配好了,即使还没有任何数据;以后对数据库中的所有访问,如把哪个表放在哪些磁道和扇区,均由数据库管理系统(如 MS SQL Server)负责。

表 2-2 列举了数据库管理系统、关系模型和操作系统中关于数据组织的一些术语,这些术语虽然用于不同领域,但经常互用。

表 2-2 术语的对应关系

数据库管理系统	关系模型	操作系统
表	关系	磁盘文件
行	元组	记录
列	属性	字段
列值	属性值	数据
列类型	域	数据类型

数据库管理系统中的核心组成部分是数据库引擎,这个部件负责解释 SQL 语句,并进行优化,然后执行,完成对数据文件的访问。应用程序(如医院的收费工作站)或者交互查询工具(如 MS SQL Server 中的 SQL 编辑器)均向数据库引擎发送 SQL 语句。SQL 在数据库管理系统中的位置如图 2-1 所示。

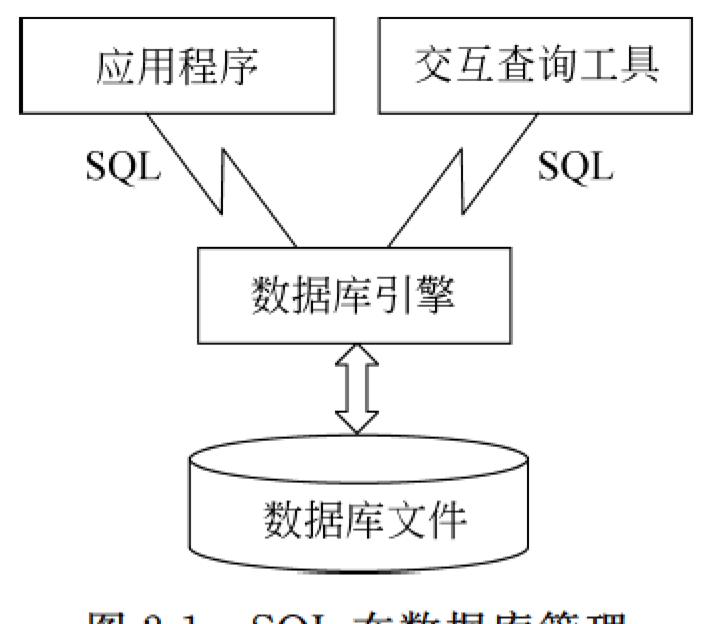


图 2-1 SQL 在数据库管理系统中的位置

SQL 有 4 个方面的功能:数据定义、数据检索、数据操纵和数据控制。

数据定义语言(Data Definition Language,DDL)是用于创建表等数据库对象的语言。命令动词有 CREATE(创建)、DROP(删除)、ALTER(修改)等。

例如,创建"医生"表的 SQL 语句如下。

```
CREATE TABLE 医生
(医生号 CHAR(6) NOT NULL,
姓名 CHAR(10) NOT NULL,
年龄 INTEGER NOT NULL,
部门 CHAR(8) NOT NULL)
```

在表“医生”上增加一列“性别”，数据类型为 CHAR(4)，完成增加列的 SQL 语句如下。

```
ALTER TABLE 医生 ADD 性别 CHAR(4)
```

删除表“医生”的 SQL 语句如下。

```
DROP TABLE 医生
```

数据检索(Data Retrieval)指从数据库中检索期望的数据。命令动词是 SELECT。例如，下面的语句从“医生”表中检索所有行，并显示这些行在所有列(*)上的值。

```
SELECT  *
FROM 医生
```

数据操纵语言(Data Manipulation Language，DML)完成插入、删除和修改 3 种操作，命令动词分别是 INSERT、DELETE 和 UPDATE。

数据控制语言(Data Control Language，DCL)的主要功能是约束数据库用户对数据的访问权限，命令动词有 GRANT 和 REVOKE 等。

2.2 基本查询

为了演示 SQL 功能，本章及后面章节使用了 SQL Server 2008 中的两个数据库：“东山县医院财务库”和“东山县医院业务库”。东山县医院财务库中有凭证库、会计科目表、科目余额表、现金日记账 101 和银行存款日记账 102。其中最主要的是凭证库。凭证库中记载了会计记账凭证。手工会计作业从凭证到日记账、明细账、总账均有不同人员登录，并定期核对，以降低出错的可能性。而电算系统的所有数据都来源于凭证，只要一次输入原始数据，系统就会自动进行后期处理。会计业务中的凭证在凭证库中使用“源凭证号”进行唯一标识。一个凭证一般与凭证库中的多行对应，即凭证库中可能有多行中的“源凭证号”是相同的。

东山县医院业务库中主要有“门诊收费”和“住院收费”的收费数据。图 2-2 展示了东山县医院业务库中表之间的关系。从图 2-2 中可以看到，“门诊收费”表和“住院收费”表只是描述了诊疗机构一次向患者收费的概况，具体收费项目则分别记录在“门诊收费明细”和“住院收费明细”中。其他表，如部门、医生、药品、检查项目都是收费记录引用的表。

“门诊收费明细”和“住院收费明细”中的项目编号既可以是“药品”中的，也可以是“检查项目”中的。

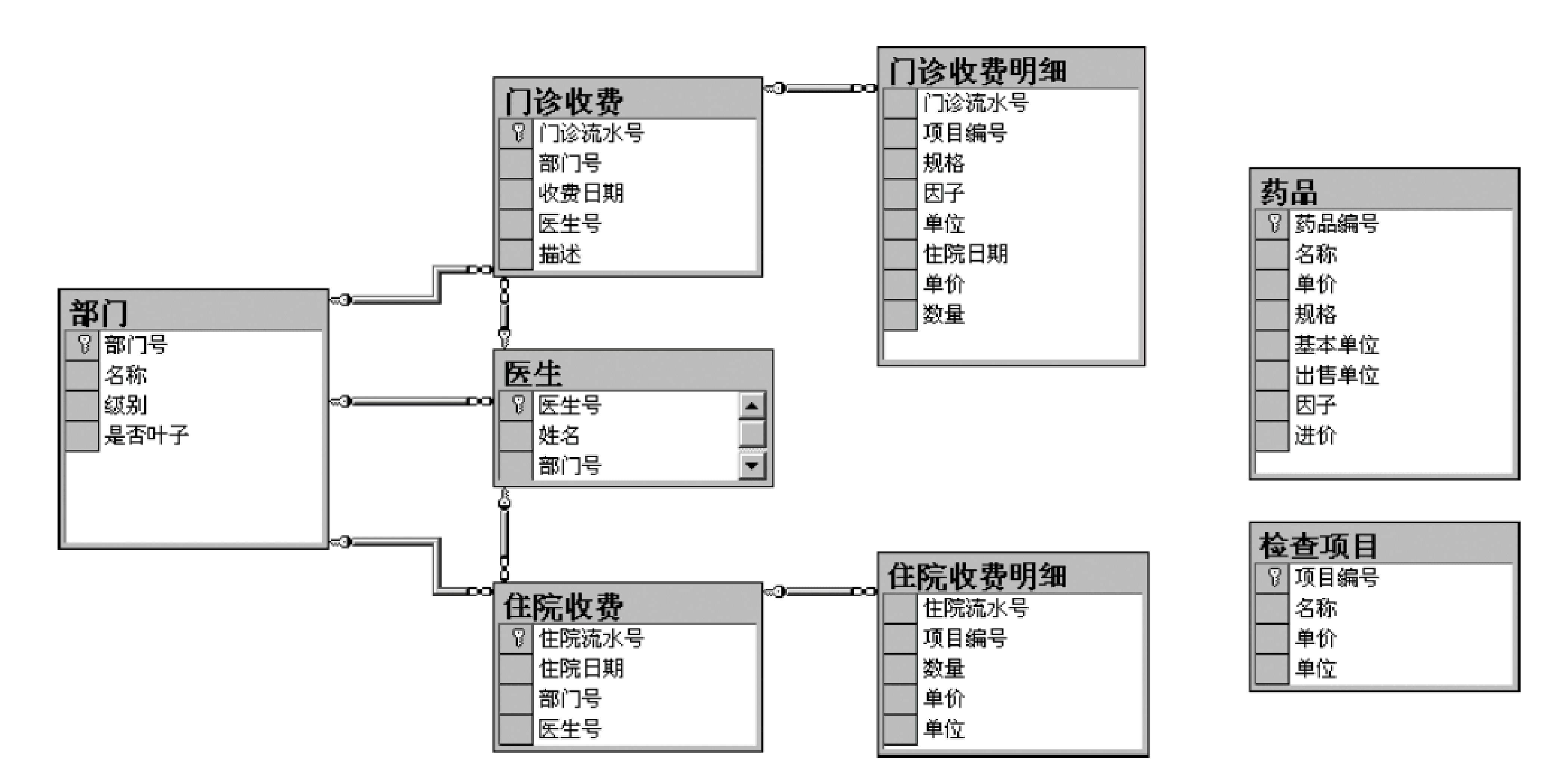


图 2-2　东山县医院业务库中表之间的关系

2.2.1　在 Management Studio 中设计和执行查询

Microsoft SQL Server Management Studio（以下称为“管理器”）包含了对象资源管理器、SQL 编辑器等客户端工具。

对象资源管理器一般位于窗口左侧，显示了数据库、表和列等对象。该管理器对于审计人员观察数据库中有哪些表、表中有哪些列非常方便。当对象资源管理器被关掉后，单击“视图”菜单中的“对象资源管理器”就可重新打开。

可以在 SQL 编辑器中执行所有语句或只执行选定的语句。通过在编辑器窗格中创建或打开脚本并单击“执行查询”按钮执行编辑器中的所有 SQL 语句；通过选择编辑器窗格中的部分代码行并单击“执行查询”按钮只执行选定的 SQL 语句。

例如，要从“东山县医院财务库”中检索源凭证号是“2003-3-31-记-41”的行，可在查询窗口中输入如下语句。

```
SELECT 凭证号,摘要
FROM  凭证库
WHERE 源凭证号='2003-3-31-记-41'
```

输入完毕后，单击“执行查询”按钮（绿色右三角）或单击“执行”按钮，查询结果显示在结果窗格中，如图 2-3 所示。

默认情况下，在编辑器窗格中不同的语法成分使用不同的文本颜色，根据代码颜色可以判断错误。例如，如果输入一个关键字 SLELCT，而它不以蓝色显示，则该关键字可能拼错了。如果许多代码都以红色显示，那么可能遗漏了字符串右边的引号。单击“分析”按钮只分析语法，而不执行语句。在结果窗格中双击错误信息，可以在代码中定位产生该错误的行。

SQL 对字母的大小写不敏感。下面语句与图 2-3 中的语句等价。

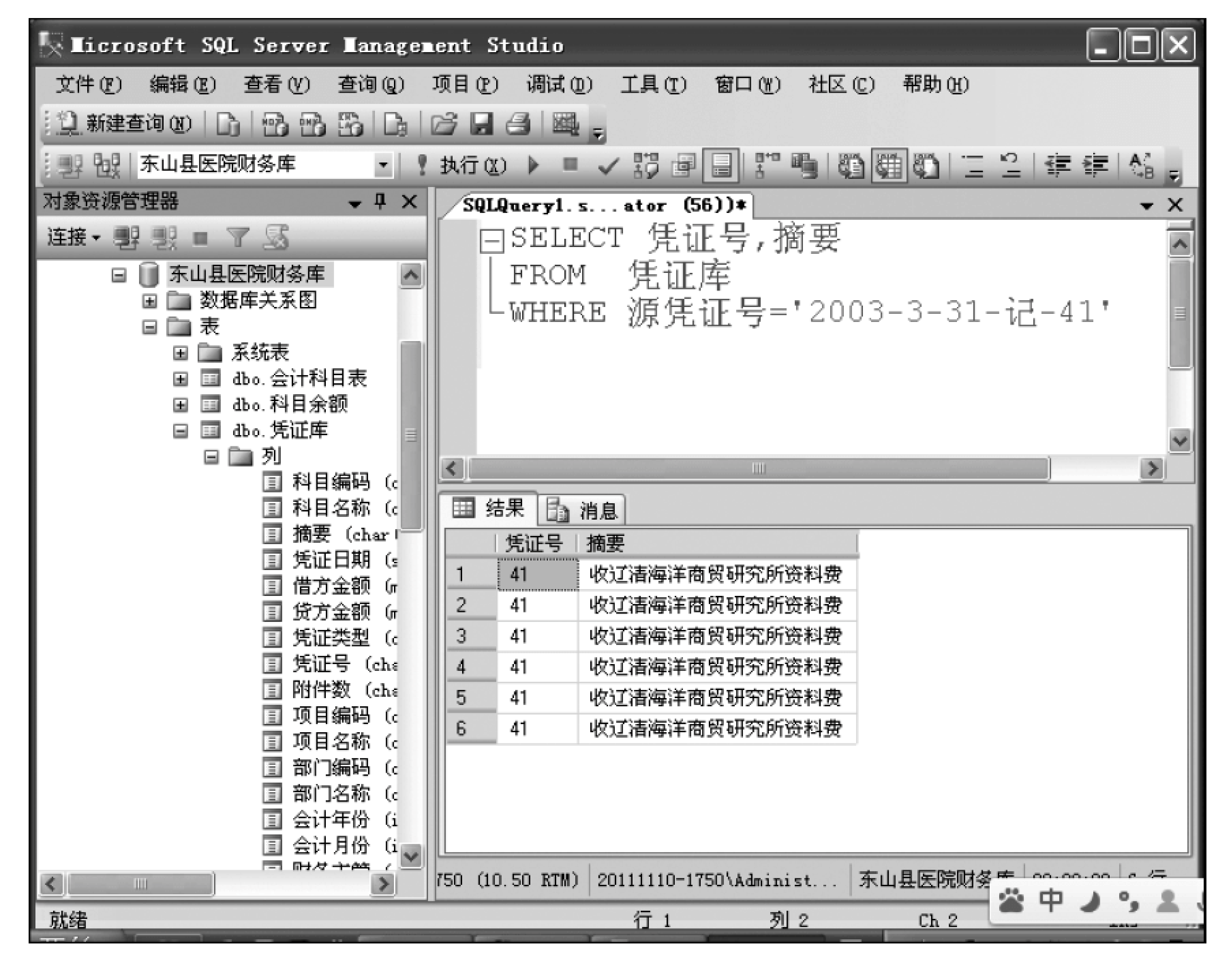


图 2-3 结果窗格

```
select 凭证号,摘要
from 凭证库
where 源凭证号='2003-3-31-记-41'
```

虽然关键字、表名、列名等大小写的效果相同，但是通常约定关键字为大写，表名、列名为小写。例如，一般写成：

```
SELECT a,b
FROM  t
WHERE a=2
```

虽然可以把各个子句写在同一行上，但是通常约定每行只写一个子句。例如，写成：

```
SELECT 凭证号,摘要
FROM  凭证库
WHERE 源凭证号='2003-3-31-记-41'
```

而不写成：

```
SELECT 凭证号,摘要 FROM  凭证库 WHERE 源凭证号='2003-3-31-记-41'
```

不同语法成分间要用空格隔开；使用空格的个数不限。相同语法成分，如列名，要用西文逗号隔开。例如：

```
SELECT 凭证号,摘要
```

```
FROM  凭证库
WHERE 源凭证号='2003-3-31-记-41'
```

等同于逗号后面增加一个空格、等于号前后各一个空格的语句。

```
SELECT 凭证号,  摘要
FROM  凭证库
WHERE 源凭证号  =  '2003-3-31-记-41'
```

除非要录入汉字，其余一律使用西文输入状态。汉字录入法不限。如果自己录入汉字的速度较慢，可以使用拖放的方法把对象资源管理器中的表名或者列名拖放到 SQL 编辑器中。

当鼠标插入点在查询窗口的查询语句窗格中时，单击工具栏中的“保存查询”按钮，在弹出的对话框中选择文件位置并命名“查询 1”可以把查询语句保存为“查询 1. sql”。注意，文件的扩展名为. sql。这个文件称为 SQL 脚本文件。

单击工具栏中的“打开文件”按钮，在弹出的对话框中找到需要的文件，如“查询 1. sql”，将其选中，单击“打开”按钮，则在查询窗口中出现前面保存过的 SQL 脚本。

在结果窗格的任意处单击，然后选择“文件”菜单下的“将结果另存为”菜单项，在弹出的对话框中选择文件位置并命名“查询 1”，可以把查询结果保存为“查询 1. CSV”。注意，文件的扩展名为. CSV。

2.2.2 基本的查询语句

最基本、最常用的 SELECT 语句语法是：

```
SELECT  <列名 1>, <列名 2>, …, <列名 n>
FROM  <表名>
WHERE  <条件表达式>
```

下面通过几个例子说明 SELECT 语句的用法。注意，列名之间使用西文的逗号隔开。

例 1 写出从凭证库表中查询所有行在“凭证号”和“摘要”列上的值的 SQL 语句。

```
SELECT 凭证号,摘要
FROM 凭证库
```

例 2 写出从凭证库表中查询所有行在所有列上的值的 SQL 语句。

```
SELECT *
FROM 凭证库
```

这里的 * 表示所有列。当然，也可以在 SELECT 后面列出所有列名。

例 3 写出从凭证库表中查询所有行在“摘要”和“凭证号”列上的值的 SQL 语句。

```
SELECT 摘要,凭证号
FROM 凭证库
```

例 4　设计 SQL 语句，从凭证库表中查询所有行在“摘要”列上的值。

```
SELECT 摘要
FROM 凭证库
```

例 5　设计 SQL 语句，从凭证库表中查询凭证号等于“1”的行在凭证号和摘要列上的值。

```
SELECT 凭证号,摘要
FROM 凭证库
WHERE 凭证号='1'
```

例 6　设计 SQL 语句，从凭证库表中查询该表中涉及哪几个会计科目。

如果从“凭证库”表中查询所有行在“科目编码”上的值：

```
SELECT 科目编码
FROM 凭证库
```

会发现，上述查询结果中出现了重复的行，即科目编码重复出现。消除结果中重复的数据行，才是需要的结果。这种情况下使用 DISTINCT。

```
SELECT  DISTINCT 科目编码
FROM 凭证库
```

DISTINCT 用于从 SELECT 语句的结果集中除去重复的行。如果没有指定 DISTINCT，查询结果可能出现重复的行。

注意，对于 DISTINCT 关键字来说，各空值将被认为是相互重复的内容。当 SELECT 语句中包括 DISTINCT 时，不论遇到多少个空值，在结果中只返回一个 NULL。DISTINCT 要写在 SELECT 关键字和第一个字段之间。

如果在 SQL 编辑器中结束了一条 SELECT 语句后，并不想删除或者修改该语句，而是保留这条语句，继续设计下一条语句并执行，一般采取通过配对使用/*和*/把不想执行的语句括起来的办法实现。单击“执行查询”按钮后，SQL 编辑器只会执行未被/*和*/括起来的部分，而忽略被/*和*/括起来的部分。

2.3　区分不同的数据类型

文史类专业的审计师通常会把所有的数据都理解为文本，但事实上，在数据库中区分各种各样的数据：有的是整数，有的是实数，有的是字符，还有的是日期或者时间。SQL Server 中常见的数据类型有精确数字、近似数字、字符串、日期和时间、货币等。表中的每个列都有数据类型。数据类型定义了一个域（domain），以及在这个域上允许的运算。数据类型实际上是对数据的一种约束，不同的数据类型可以进行不同的运算。

精确数字有整数、比特、小数和货币 4 种类型。

整数有 bigint、int、smallint、tinyint 4 种。bigint 类型是从 -2^{63}（−9 223 372 036 854 775 808）到 $2^{63}-1$（9 223 372 036 854 775 807）的整型数据（所有数字），使用 8 字节存储；int 类型

是从-2^{31}（-2 147 483 648）到$2^{31}-1$（2 147 483 647）的整型数据（所有数字），使用4字节存储，例如“凭证库”表中的“会计年份”列的类型为int；smallint类型是从-2^{15}（-32 768）到$2^{15}-1$（32 767）的整数数据，使用2字节存储，例如“会计科目”表中的“部门核算”列的类型为smallint；tinyint是0～255的整数数据，使用1字节存储，例如“部门”表的“级别”列为tinyint。

比特（bit）类型是仅有1或0两个取值的精确数字，如“部门”表“是否叶子”列的类型为比特类型。

小数类型有decimal和numeric两种。decimal是从$-10^{38}+1$到$10^{38}-1$的固定整数部分和小数部分的数字数据。numeric在功能上等同于decimal。numeric和decimal数据类型的默认最大精度值是38。精度指有效数字的位数。例如，decimal(6，2)表示小数点后有2位数字，小数点前有4位数字。

货币类型money固定为4位小数，它实际上是带有4位小数的decimal数据。货币数据值介于-2^{63}（-9 223 372 036 854 775 808）与$2^{63}-1$（+9 223 372 036 854 775 807）之间。例如，“检查项目”表的“单价”列就是货币类型。smallmoney也是货币类型，其数据值介于－214 748.3648与＋214 748.3647之间。如果数值超过了上述范围，则可使用decimal数据类型代替。money和smallmoney被限制到小数点后4位。如果要求小数点后有更多位，则使用decimal数据类型。

近似数字有float和real两种类型。

float类型表示从－1.79E＋308到1.79E＋308的浮点数字，使用8字节存储。例如，“凭证库”表中的“借方金额”列的类型为float。虽然“借方金额”类型可以为float，但一般使用money。因为在计算机内部使用二进制表示，当表示一个浮点数和进行浮点计算时，存在舍入误差。譬如，0.1就不能被精确地表示。26个0.1相加得到的是2.600000000000001，而不是2.6。两个浮点数字的差小于一个很小的数，如相差10^{-8}就可以认为两个浮点数相等。

real是从－3.40E＋38到3.40E＋38的浮点精度数字，使用4字节存储。

由于float和real数据类型的这种近似性，当要求精确的数字状态时，如在财务应用程序中，在那些需要舍入的操作中，或在等值核对的操作中，就不使用这些数据类型。这时就要用integer、decimal、money或smallmoney数据类型。

非Unicode编码字符（char、varchar、text），如果是英文，则使用1字节存储；如果是汉字，则使用2字节存储。char(n)是固定长度的非Unicode字符数据，最大长度为8000个字符；varchar(n)是可变长度的非Unicode数据，最长为8000个字符；text是可变长度的非Unicode数据，最大长度为$2^{31}-1$（2 147 483 647）个字符。

对于Unicode编码字符串（nchar、nvarchar、ntext），字符（如中、日、韩等）均使用2字节编码。nchar(n)表示固定长度的Unicode数据，最大长度为4000个字符；nvarchar(n)表示可变长度Unicode数据，其最大长度为4000个字符；ntext表示可变长度Unicode数据，其最大长度为$2^{30}-1$（1 073 741 823）个字符。Unicode为每一个字符提供一个唯一的编码（即一组数字）。Unicode编码系统可分为编码方式和实现方式两种。一个字符的Unicode编码是确定的，但对Unicode编码的实现方式有所不同。Unicode的实现方式称

为 Unicode 转换格式(Unicode Translation Format，UTF)。

如果需要跨平台传输字符，建议使用 Unicode 编码字符串。使用 Unicode 编码的目的是解决不同计算机系统、不同应用系统中处理各国文字的问题。使用 nchar 或者 nvarchar 可以减少字符转换问题，防止在某些情况下出现乱码。

如果所存数据长度基本一致，建议使用 char 型或者 nchar 型，这样查询速度快，但是使用时注意剔除数据两边的空格；如果所存数据长度差异较大，可以使用 varchar 型或者 nvarchar 型。

常用的日期和时间数据类型有 datatime 和 smalldatetime 两种。datetime 数据类型表示从 1753 年 1 月 1 日到 9999 年 12 月 31 日的日期和时间数据，精确到百分之三秒(或 3.33ms)，使用 8 字节存储，如"门诊收费明细"表中的"住院日期"列；smalldatetime 数据类型表示从 1900 年 1 月 1 日到 2079 年 6 月 6 日的日期和时间数据，精确到分钟，使用 4 字节存储，如"凭证库"表中的"凭证日期"列。单独的日期类型和时间类型分别是 data 和 time。

2.4 字面量的格式要求

字面量也称为字面值或常量，是在 SQL 脚本中表示特定数据值的符号。例如，查询凭证日期为 2013 年 3 月 11 日的所有凭证的 SQL 语句中就使用了字面量 2013-03-11：

```
SELECT  *
FROM 凭证库
WHERE 凭证日期 = '2013-03-11'
```

在 SQL 语句中，不同类型的字面量有不同的格式要求。例如，2013-03-11 就是日期型字面量，要求使用单引号，并使用"-"分隔年、月、日。

字面量的格式取决于它所表示的值的数据类型。根据数据类型的不同，有字符串字面量、比特字面量、日期和时间字面量、整数字面量、小数字面量、浮点数和实数字面量、货币字面量等。

字符串字面量使用单引号引起来，字符串中可以包含字母、数字字符(a～z、A～Z 和 0～9)以及特殊字符，如感叹号 (!)、at 符 (@) 和数字号 (#)等。例如，'河北省审计厅'，'文件复制完成了 50% .' 等。

中间没有任何字符的两个单引号'' 表示"空"字符串。

比特(bit)字面量使用数字 0 或 1 表示，并且不使用引号。

日期和时间(datetime)字面量使用特定格式的字符日期值表示，并被单引号括起来。例如：'2017-12-01 14:05:00 '表示"2017 年 12 月 1 日下午 2 点 5 分 0 秒"。

以下格式的日期字面量也是合法的：'April 15, 1998'，'15 April, 1998'，'980415'，'20080101'，'04/15/1998'，'2006-10-15'，'2006/10/15'；时间字面量'14:30:24' 表示 24 小时格式的下午 2 点 30 分 24 秒、'04:24 PM'表示 12 小时格式的下午 4 点 24 分、'04:24 AM'表示 12 小时格式的凌晨 4 点 24 分。

整数(integer)字面量由没有用引号括起来且不含小数点的一串数字表示。例如12。

小数(decimal)字面量由没有用引号括起来且包含小数点的一串数字表示。例如12.3。

浮点数(float)和实数(real)字面量使用科学计数法表示,如101.5E5、0.5E−2。

货币(money)字面量表示以可选小数点和可选货币符号作为前缀的一串数字。这些字面量不使用引号,如$542023.14。

若要指明一个数是正数还是负数,则对数字字面量应用+或−的一元运算符。使用NULL或者null表示“空值”字面量。

2.5 使用表达式

表达式是标识符、字面量和运算符的组合,并能通过计算得到一个值。SQL语句中可在多个不同的位置出现表达式。下面通过几个例子演示如何在SQL语句中使用表达式。

例7 设计SQL语句,从药品表中查询所有药品,显示其编号和加价10%后的单价。

```
SELECT 药品编号, 单价 * 1.1
FROM 药品
```

在这个例子中,“药品编号”本身是表达式,它返回“药品编号”列中的值;单价 * 1.1是算术表达式,这个表达式用作查询结果中的列。

例8 设计SQL语句,从凭证库表中查询“102”科目的摘要。

```
SELECT 摘要
FROM 凭证库
WHERE 科目编码 ='102'
```

在这个例子中,科目编码 = '102' 也是表达式,它返回逻辑值“真”或者“假”,称为布尔表达式。将表达式用作查询条件,可以利用该表达式检索期望的行。

例9 如果查询结果中不涉及表中的列,则可以没有FROM子句,例如:

```
SELECT  '河北省审计厅', '培训', 2007
```

查询结果如下(结果仍为一个表)。

```
(无列名)      (无列名)      (无列名)
河北省审计厅   培训    2007
```

运算符用来执行算术、比较、连接或赋值操作。SQL Server有7类运算符,见表2-3。

表2-3 运算符汇总

运 算 符	执 行 运 算
算术	加法、减法、乘法、除法、求模
赋值	将值赋给变量,或将结果集列与别名相关联

续表

运 算 符	执 行 运 算
比较	将数值与另一个数值或表达式进行比较
逻辑	真假，如 AND、OR、NOT
谓词	返回布尔值，常见的有 LIKE、ANY、ALL、IN 等
字符串串联	将两个字符串(字符或二进制数)合并为一个字符串
一元	执行有一个操作数的运算，如正、负

算术运算符包括加、减、乘、除、取模等。这些运算的符号及含义见表 2-4。

表 2-4　算术运算符

运算符	名称	含　　义
+	加	加法
−	减	减法
*	乘	乘法
/	除	除法
%	模	返回一个除法的整数余数。例如，12%5=2，这是因为 12 除以 5 的余数为 2

加(+)和减(−)运算符也可用于对 datetime 及 smalldatetime 值执行算术运算。使用()可以改变运算符优先级：2*(6+9)/3。

例 10　设计 SQL 语句，从药品表中查询所有药品的名称以及增加一元后的药品的单价。

```
SELECT 名称, 单价 +1
FROM 药品
```

此例中使用加法运算符计算“单价”，在查询结果中形成一个新的列。注意，表中的“单价”列并未发生变化，仅在查询结果中单价增加了 1。但是，由表达式“单价+1”形成的查询结果的新列并没有列名。在 SELECT 语句中，如果某列是一个表达式，则该列默认没有列名。以下语句为表达式“单价+1”指定了名字“新单价”。

```
SELECT 名称, 单价 +1  AS  新单价
FROM 药品
```

当然，列名可以直接是表达式，也可以为表达式指定一个列名，如：

```
SELECT 名称 AS 药品名称, 单价 +1  AS 新单价
FROM 药品
```

这个 SELECT 语句中为“名称”列指定了另外一个列名“药品名称”。通过保留字 AS 指定的列名称为列的“别名(alias)”。

续表

运　算　符	执行运算
比较	将数值与另一个数值或表达式进行比较
逻辑	真假，如 AND、OR、NOT
谓词	返回布尔值，常见的有 LIKE、ANY、ALL、IN 等
字符串串联	将两个字符串（字符或二进制数）合并为一个字符串
一元	执行有一个操作数的运算，如正、负

算术运算符包括加、减、乘、除、取模等。这些运算的符号及含义见表 2-4。

表 2-4　算术运算符

运算符	名称	含　　义
＋	加	加法
－	减	减法
*	乘	乘法
/	除	除法
%	模	返回一个除法的整数余数。例如，12%5＝2，这是因为 12 除以 5 的余数为 2

加（＋）和减（－）运算符也可用于对 datetime 及 smalldatetime 值执行算术运算。使用()可以改变运算符优先级：2 * (6＋9)/3。

例 10　设计 SQL 语句，从药品表中查询所有药品的名称以及增加一元后的药品的单价。

```
SELECT 名称, 单价 +1
FROM 药品
```

此例中使用加法运算符计算“单价”，在查询结果中形成一个新的列。注意，表中的“单价”列并未发生变化，仅在查询结果中单价增加了 1。但是，由表达式“单价＋1”形成的查询结果的新列并没有列名。在 SELECT 语句中，如果某列是一个表达式，则该列默认没有列名。以下语句为表达式“单价＋1”指定了名字“新单价”。

```
SELECT 名称, 单价 +1  AS  新单价
FROM 药品
```

当然，列名可以直接是表达式，也可以为表达式指定一个列名，如：

```
SELECT 名称 AS 药品名称, 单价 +1  AS 新单价
FROM 药品
```

这个 SELECT 语句中为“名称”列指定了另外一个列名“药品名称”。通过保留字 AS 指定的列名称为列的“别名(alias)”。

加法算术运算符也可以将一个以天为单位的数字加到日期中。例如,从凭证库表中查询所有的凭证日期和该凭证日期的次日。

```
SELECT 凭证日期,凭证日期 +1 AS 次日
FROM 凭证库
```

“次日”就是通过把 datetime 类型的凭证日期加 1 实现的。

例 11 设计 SQL 语句,从药品表中汇总查询所有药品的名称、单价与进价之差。

```
SELECT 名称,单价 -进价 AS 差价
FROM 药品
```

这个例子使用减法运算符计算所有药品零售价与进价之差。减法算术运算符也可以从日期中减去一个以天数为单位的数值。例如,从凭证库表中查询所有的凭证日期和该凭证日期的前一日。

```
SELECT 凭证日期,凭证日期 -1
FROM 凭证库
```

这个例子从 datetime 日期类型的凭证日期减去 1 天得到该日期的前一天。

例 12 设计 SQL 语句,从药品表中查询所有药品的名称以及加价 50%后的单价。

```
SELECT 名称,单价 * 1.5 AS 新单价
FROM 药品
```

这个例子通过使用算术乘法运算符的表达式得到加价 50%后的结果。

例 13 设计 SQL 语句,从药品表中查询所有药品名称及其单价的半价。

```
SELECT 名称,单价 /2 AS 半价
FROM 药品
```

这个例子使用算术除法运算符用“单价”除以 2 得到“半价”。

如果算术表达式中出现 NULL 值,则结果仍为 NULL。例如,SELECT5+NULL 的结果为 NULL。

如果两个操作数之一是字符串,那么加号就成为字符串串联运算符。

例 14 设计 SQL 语句,从医生表中查询所有医生的姓名和医生号,并以“姓名(医生号)”格式显示查询结果。

```
SELECT 姓名 +'(' +医生号 +')'
FROM 医生
```

表达式“姓名 + '(' + 医生号 + ')'”把 4 个字符串连接在一起:姓名、(、医生号、)。假如姓名是“陶继民”,医生号是“20101”,那么查询结果为“陶继民(20101)”。

比较运算符的结果是布尔数据类型,它有 3 种值:TRUE、FALSE 及 UNKNOWN。比较运算符见表 2-5。

表 2-5　比较运算符

运算符	含　　义	运算符	含　　义
=	等于	<>	不等于
>	大于	!=	不等于(非 SQL-92 标准)
<	小于	!<	不小于(非 SQL-92 标准)
>=	大于或等于	!>	不大于(非 SQL-92 标准)
<=	小于或等于		

下面通过几个例子演示比较运算符的使用。

例 15　设计 SQL 语句，从凭证库表中查询科目编码为 102 的行。

```
SELECT *
FROM 凭证库
WHERE 科目编码 = '102'
```

这个例子使用比较运算符“=”，比较凭证库表中每一行的科目编码是否与“102”相等。由于科目编码是 char 类型，所以字面量“102”要用单引号引起来。

例 16　设计 SQL 语句，从凭证库表中查询借方金额大于 10000 元的行。

```
SELECT *
FROM 凭证库
WHERE 借方金额 >10000
```

这个例子使用比较运算符“>”判断凭证库表中每一行的借方金额是否大于 10000 元。

例 17　设计 SQL 语句，从凭证库表中查询借方金额大于或等于 10000 元的行。

```
SELECT *
FROM 凭证库
WHERE 借方金额 >=10000
```

这个例子使用比较运算符“>=”判断凭证库表中每一行的借方金额是否大于或等于 10000 元。

例 18　设计 SQL 语句，从凭证库表中查询借方金额小于 10000 元的行。

```
SELECT *
FROM 凭证库
WHERE 借方金额 <10000
```

这个例子使用比较运算符“<”判断凭证库表中每一行的借方金额是否小于 10000 元。

例 19　设计 SQL 语句，从凭证库表中查询借方金额小于或等于 10000 元的行。

```
SELECT *
FROM 凭证库
WHERE 借方金额 <=10000
```

这个例子使用比较运算符“<=”判断凭证库表中每一行的借方金额是否小于或等于10000元。

例 20 设计 SQL 语句，从凭证库表中查询科目名称不是“银行存款”的行。

```
SELECT *
FROM 凭证库
WHERE 科目名称 <> '银行存款'
```

这个例子使用比较运算符“<>”判断凭证库表的每一行的科目名称是否为“银行存款”。

例 21 设计 SQL 语句，从会计科目表中查询顶级科目。

```
SELECT *
FROM 会计科目表
WHERE 上级科目编码 IS NULL
```

“顶级科目”即没有上级科目的科目，也就是该科目的“上级科目编码”为空。这个例子使用保留字 is 判断会计科目表的每一行，也就是每个科目的“上级科目编码”是否为空，保留字 NULL 表示“空值”。注意，NULL 的含义是没有值，而不是值为 0 或空格。不要使用比较运算符“=”与 NULL 进行比较，如：

```
SELECT *
FROM 会计科目表
WHERE 上级科目编码 =NULL
```

这是因为“上级科目编码 = NULL”的结果是 UNKNOWN(不知道)，NULL=NULL 的比较结果默认也是 UNKNOWN。

例 22 设计 SQL 语句，从会计科目表中查询非顶级科目。

```
SELECT *
FROM 会计科目表
WHERE 上级科目编码 IS NOT NULL
```

“非顶级科目”即上级科目编码不为空的科目。这个例子中，IS NOT NULL 表示“不为空”。

逻辑运算符对某个条件进行测试，以获得其真实情况。逻辑运算符和比较运算符一样，都返回带有 TRUE 或 FALSE 值的布尔数据类型。T-SQL 中的逻辑运算符见表 2-6。

表 2-6 T-SQL 中的逻辑运算符

运算符	名称	含　　义
AND	而且	如果其两侧的条件同时成立，则运算结果成立(TRUE)
NOT	否	对任何其他布尔运算符的值取反
OR	或	如果其两侧的条件有一个成立，则运算结果就为 TRUE

这 3 个逻辑运算符中，NOT 优先级最高，OR 优先级最低。比较运算符优先级高于逻辑运算符。下面通过几个例子演示逻辑运算符的使用。

例 23 设计 SQL 语句，从凭证库表中查找满足两个条件的行：一是科目名称为“现金”；二是贷方金额为 10000 元以上（含 10000 元）。

```
SELECT *
FROM 凭证库
WHERE 科目名称 = '现金' AND 贷方金额 >= 10000
```

在这个例子中，关系表达式“科目名称 = '现金'”表达了第 1 个条件，关系表达式“贷方金额 >= 10000”表达了第二个条件，由于要求这两个条件都满足，所以使用逻辑运算符 AND 进行连接。AND 运算符要求 AND 前后的表达式都为真时，AND 运算的结果才为真。数据库引擎根据逻辑表达式“科目名称 = '现金' AND 贷方金额 >= 10000”逐行扫描凭证库，如果第某行的科目名称等于“现金”，而且第 i 行的贷方金额大于或等于 10000 元，则将该行作为结果集中的一行。

例 24 设计 SQL 语句，从凭证库表中查询满足两个条件之一的行：贷方金额 10000 元以上（含 10000 元），或借方金额 10000 元以上（含 10000 元）。

```
SELECT *
FROM 凭证库
WHERE 贷方金额 >= 10000 OR 借方金额 >= 10000
```

在这个例子中，使用关系表达式“贷方金额>=10000”表达前一个条件；使用关系表达式“借方金额>=10000”表达后一个条件，由于要求这两个条件之一满足即可，所以使用逻辑运算符 OR 进行连接。OR 运算符连接的两个条件只要其中一个条件为真，OR 运算的结果就为真。数据库引擎根据逻辑表达式“贷方金额>=10000 OR 借方金额>=10000”逐行扫描凭证库，如果某行的贷方金额大于或等于 10000 元，或者借方金额大于或等于 10000 元，则将该行作为结果集中的一行。

例 25 设计 SQL 语句，从凭证库表中查询满足以下两个条件的行：科目名称为“现金”，而且贷方金额或借方金额为 10000 元以上（含 10000 元）。

```
SELECT *
FROM 凭证库
WHERE 科目名称 = '现金' AND (贷方金额 >= 10000 OR 借方金额 >= 10000)
```

这个例子使用关系表达式“科目名称 = '现金'”表达第一个条件；使用逻辑表达式“贷方金额 >= 10000 OR 借方金额 >= 10000”表达第二个条件，使用逻辑运算符 AND 表示“而且”。如果把 WHERE 后面的条件写成“科目名称 = '现金' AND 贷方金额 >= 10000 OR 借方金额 >= 10000”，那么其含义就变成：科目名称是“现金”而且贷方金额为 10000 元以上，或者借方金额为 10000 元以上（无论科目名称是否是“现金”）。这是因为逻辑运算符 OR 的优先级低于逻辑运算符 AND，数据库引擎就会先计算“科目名称 = '现金' AND 贷方金额 >= 10000”，然后再与关系表达式“借方金额 >= 10000”进行 OR 运算。

例 26 设计 SQL 语句，从凭证库表中查询科目名称不是"现金"的行。

```
SELECT *
FROM 凭证库
WHERE  NOT 科目名称 = '现金'
```

在这个例子中，数据库引擎会先计算关系表达式"科目名称 = '现金'"，得到一个逻辑值，然后再使用逻辑运算符 NOT 改变这个逻辑值。

例 27 设计 SQL 语句，从凭证库中查找满足这两个条件的行：科目名称不是"现金"，而且贷方金额或借方金额为 10000 元以上。

```
SELECT *
FROM 凭证库
WHERE NOT 科目名称 = '现金' AND (贷方金额 >= 10000 OR 借方金额 >= 10000)
```

在这个例子中，使用逻辑表达式"NOT 科目名称 = '现金'"表达了第一个条件；使用逻辑表达式"贷方金额 >= 10000 OR 借方金额 >= 10000"表达了第二个条件。这两个条件使用逻辑运算符 AND 连接。

例 28 设计 SQL 语句，从凭证库表中查询科目名称为"银行存款"或者"现金"的行。

```
SELECT *
FROM 凭证库
WHERE 科目名称 = '现金' OR 科目名称 = '银行存款'
```

这个例子使用关系表达式"科目名称 = '现金'"表达了科目名称为"现金"，使用关系表达式"科目名称 = '银行存款'"表达了科目名称为"银行存款"。由于要求这两个条件二者之一满足即可，所以使用逻辑运算符 OR 连接。注意，不要写成：

```
SELECT *
FROM 凭证库
WHERE 科目名称 = '现金' OR '银行存款'
```

这是因为表达式"'银行存款'"不是一个逻辑表达式。

除了以上介绍的算术运算符、关系运算符、逻辑运算符外，还有一类运算符称为"谓词"运算符。这些运算符使用保留字标识。谓词运算的结果为一个逻辑值。常见的谓词运算符及其含义见表 2-7。谓词运算符的优先级高于逻辑运算符。

表 2-7 常见的谓词运算符及其含义

运算符	含　义
IN	如果操作数等于表达式列表中的一个，那么就为 TRUE
BETWEEN…AND	如果操作数在某个范围之内，那么就为 TRUE
LIKE	如果操作数与一种模式相匹配，那么就为 TRUE
EXISTS	如果子查询不为空，那么就为 TRUE
ALL	如果一系列的比较都为 TRUE，那么就为 TRUE

续表

运算符	含　　义
ANY	如果一系列的比较中任何一个为 TRUE，那么就为 TRUE
SOME	如果在一系列比较中有些为 TRUE，那么就为 TRUE。与 ANY 等价

下面通过几个例子说明谓词运算符的应用。

例 29 从凭证库表中查询科目名称为“银行存款”，或者“现金”，或者“应交营业税”的行。

```
SELECT *
FROM 凭证库
WHERE 科目名称 = '银行存款' OR 科目名称 = '现金' OR 科目名称 = '应交营业税'
```

这个例子要求科目名称满足 3 个取值之一即可，使用逻辑运算符 OR 连接 3 个关系表达式能够表达科目名称为“银行存款”，或者“现金”，或者“应交营业税”，但是不够简洁。这种情况下可以使用谓词 IN。

```
SELECT *
FROM 凭证库
WHERE 科目名称 IN ('银行存款', '现金', '应交营业税')
```

表达式“科目名称 IN ('银行存款', '现金', '应交营业税')”与表达式“科目名称 = '银行存款' OR 科目名称 = '现金' OR 科目名称 = '应交营业税”具有相同的含义。

例 30 从凭证库表中查询借方金额大于或等于 10000 元同时小于或等于 20000 元的行。

```
SELECT *
FROM 凭证库
WHERE 借方金额 >= 10000 AND 借方金额 <= 20000
```

这个例子使用关系表达式“借方金额 >= 10000”表达“借方金额大于或等于 10000 元”；使用关系表达式“借方金额 <= 20000”表达“借方金额小于或等于 20000 元”，然后使用逻辑运算符 AND 连接这两个条件，表示需要“同时”满足。由于这两个条件都是对“借方金额”的限制，而且限制在一个闭区间内，所以这种情况下可使用“BETWEEN AND”。

```
SELECT *
FROM 凭证库
WHERE 借方金额 BETWEEN 10000 AND 20000
```

表达式“借方金额 BETWEEN 10000 AND 20000”的含义与表达式“借方金额 >= 10000 AND 借方金额 <= 20000”的含义相同。

例 31 设计 SQL 语句，从凭证库表中查询借方金额为 10000～20000 元，而且科目名称为“现金”的行。

```
SELECT *
```

```
FROM 凭证库
WHERE 借方金额 BETWEEN 10000 AND 20000 AND 科目名称 = '现金'
```

从这个例子中可以看到，使用 BETWEEN AND 谓词使得表达式更加简洁、易懂。

谓词 LIKE 与通配符配合，用于模糊查询。通配符有两个，即%和_。其中，%通配任意字符串，包括空串；而_仅通配单个字符，不包括空串。每个汉字或每个英文字母都是一个字符。下面通过几个例子说明如何应用谓词 LIKE。

例 32 设计 SQL 语句，从凭证库表中查询摘要包含“费”的行。

```
SELECT * FROM 凭证库 WHERE 摘要 LIKE '%费%'
```

表达式“%费%”的意思是“费”字的前边有 0 个或若干个字符，后边也有 0 个或若干个字符。表达式“摘要 LIKE '%费%'”的含义是：“摘要”字符串的模式像“%费%”，如“收培训费”“会务费”“费用”“收辽清海洋商贸研究所资料费”“提取差旅费”等都是满足模式“%费%”的字符串。

例 33 设计 SQL 语句，从凭证库表中查询摘要以“费”结束的行。

```
SELECT * FROM 凭证库 WHERE 摘要 LIKE '%费'
```

例 34 设计 SQL 语句，从凭证库表中查询摘要以“会议”开始的行。

```
SELECT * FROM 凭证库 WHERE 摘要 LIKE '会议%'
```

例 35 设计 SQL 语句，从凭证库表中查询摘要的第二个字是“费”的行。

```
SELECT * FROM 凭证库 WHERE 摘要 LIKE '_费%'
```

这个例子限制“费”字必须是字符串中的第二个字，即“费”字的前面有且仅有一个字符，而不管这个字符是什么。模式“_费%”表达了这个含义。

例 36 设计 SQL 语句，从凭证库表中查询摘要倒数第三个字是“费”的行。

```
SELECT * FROM 凭证库 WHERE 摘要 LIKE '%费__'
```

例 37 设计 SQL 语句，从会计科目表中查询科目编码是 504 开头的会计科目。

```
SELECT * FROM 会计科目表 WHERE 科目编码 LIKE '504%'
```

当一个复杂的表达式有多个运算符时，运算符优先级决定执行运算的先后次序。运算符的优先级见表 2-8。沿箭头方向，优先级降低。在较低等级的运算符之前先对较高等级的运算符求值。

当一个表达式中的两个运算符有相同的运算符优先等级时，基于它们在表达式中的位置对其从左到右进行求值。例如，在表达式 5－2＋27 中，在加号运算符之前先对减号运算符进行求值。

在表达式中可以使用括号改变所定义的运算符的优先性。首先对括号中的内容进行求值，从而产生一个值，然后括号外的运算符才可以使用这个值。例如，在表达式 2＊4＋5 中，乘运算符比加运算符有更高的优先权，所以先对乘运算符进行求值，表达式的结果是 13。在表达式 2＊(4＋5)中，括号使得加法首先进行运算，整个表达式结果是 18。

表 2-8　运算符的优先级

运　算　符	优先级
＋(正)、－(负)	↓
*(乘)、/(除)、%(模)	
＋(加)、＋(串联)、－(减)	
=、>、<、>=、<=、<>、!=、!>、!<比较运算符	
BETWEEN…AND、IN、LIKE、SOME、EXISTS、ALL、ANY 谓词运算符	
NOT	
AND	
OR	

如果表达式有嵌套的括号，就先对嵌套最深的表达式求值。表达式 2*(4+(5－3))中包含嵌套的括号，其中表达式 5－3 在嵌套最深的那对括号中。该表达式产生一个值 2，然后加运算符将这个结果与 4 相加，这样就得出一个为 6 的值，之后将 6 与 2 相乘，最后表达式的结果为 12。

2.6　应用内置函数完成通用功能

T-SQL 中有 5 类内置函数：聚合函数、日期和时间函数、数学函数、字符串函数和系统函数。这些内置函数实现了一些通用的功能，如求和、求平均数等。

2.6.1　聚合函数与聚合查询

聚合函数对一组值执行计算并返回单一的值。聚合函数忽略空值。聚合函数经常与 SELECT 语句的 GROUP BY 子句一同使用构成聚合查询。常见的聚合函数见表 2-9。

表 2-9　常见的聚合函数

聚合函数	功　　能	聚合函数	功　　能
sum( )	求和	min( )	求最小值
avg( )	求平均值	stdev( )	统计标准偏差(方差的平方根)
count( )	计数	var( )	统计方差
max( )	求最大值		

下面通过几个例子说明聚合函数的用法。

例 38　设计 SQL 语句，从药品表中查询所有药品单价的平均值。

```
SELECT avg(单价)
FROM 药品
```

例 39　设计 SQL 语句，从凭证库表中查询“科目名称”为“现金”的行在“借方金额”

列上的平均值。

```
SELECT avg(借方金额)
FROM 凭证库
WHERE 科目名称 ='现金'
```

求均值函数 avg()的完整格式是：avg（[ALL | DISTINCT]<表达式>）。该函数通常用来计算某一列上的平均值，忽略其中的空值。ALL 对所有的值（无论是否重复）进行聚合函数运算，是默认设置；而 DISTINCT 只使用每个值的唯一实例。其他聚合函数中也可以使用 ALL 和 DISTINCT。

例如，表 T1 中数值型列 Num 中含有一个空值，如下所示。

T1：

Num	Name
	陈明
50	张力
70	董庆

那么执行查询语句：

```
SELECT avg(Num)
FROM  T1
```

的结果为 60。因为表中有 3 行，其中 Num 取值不为空的行有 2 行，所以平均值是(50+70)÷2=60，而不是(50+70)÷3=40。

再如，表 T2 中的 Num 上含有重复值 50，如下所示。

T2：

Num	Name
50	陈明
50	张力
70	董庆

执行语句：

```
SELECT avg(DISTINCT  Num)
FROM  T2
```

的结果为 60。这是因为保留字 DISTINCT 把重复值去掉了，按照(50+70)÷2 进行计算，而不是按照(50+50+70)÷3 进行计算。

例 40 设计 SQL 语句，从凭证库表中查询“借方金额”列上的最大值。

```
SELECT max(借方金额)
FROM 凭证库
```

例 41 设计 SQL 语句，从凭证库表中查询“借方金额”列上的最小值。

```
SELECT min(借方金额)
FROM 凭证库
```

例 42 设计 SQL 语句，从凭证库表中查询“借方金额”列上的最大值和最小值。

```
SELECT max(借方金额), min(借方金额)
FROM 凭证库
```

例 43 设计 SQL 语句，从凭证库表中查询“借方金额”列的总和。

```
SELECT sum(借方金额)  AS 借方金额汇总
FROM 凭证库
```

例 44 设计 SQL 语句，从凭证库表中查询“借方金额”“贷方金额”两个列的总和。

```
SELECT sum(借方金额) AS 借方金额汇总, sum(贷方金额) AS 贷方金额汇总
FROM 凭证库
```

例 45 设计 SQL 语句，从住院收费表中查询有多少条收费记录。

```
SELECT count(*)
FROM 住院收费
```

例 46 设计 SQL 语句，从住院收费表中查询 10205 部门有多少条收费记录。

```
SELECT count(*)
FROM 住院收费
WHERE 部门号 ='10205'
```

count(*)返回表中行的个数，包括重复行；而 count(DISTINCT *)只统计一次重复行。

已知表 T1 和 T2 如下所示，T1 表中 Num 列上含有空值，T2 表中含有重复行。

T1：

Num	Name
	陈明
50	张力
70	董庆

T2：

Num	Name
50	张力
50	张力
70	董庆

下面几个查询语句的查询结果不同。

SELECT count(*) FROM T1 的查询结果为 3。

SELECT count(*)FROM T2 的查询结果为 3，包含重复行。

SELECT count(num) FROM T1 的查询结果为 2，这是因为默认情况下 count()忽略空值。

SELECT count(num)FROM T2 的查询结果为 3，这是因为默认统计重复值。

SELECT count(distinct num) FROM T2 的查询结果为 2，这是因为不统计重复值。

使用聚合函数的查询称为聚合查询,在没有使用 GROUP BY 子句的情况下,查询结果最多只有一行。而且,在没有使用 GROUP BY 子句的情况下,SELECT 后面的每个表达式都要使用聚合函数。例如,以下 SQL 语句是错误的。

```
SELECT 科目编码, 科目名称, min(借方金额)
FROM 凭证库
```

在 SQL 编辑器中执行,将返回以下错误信息。

```
服务器:消息 8118,级别 16,状态 1,行 1
```

列 '凭证库. 科目编码' 在选择列表中无效,因为该列未包含在聚合函数中,并且没有 GROUP BY 子句。

2.6.2　日期和时间函数

"日期"指含有年、月、日 3 个分量的数据类型。"时间"指含有时、分、秒 3 个分量的数据类型。"日期时间"指含有年、月、日、时、分、秒 6 个分量的数据类型。选择合适的面向日期和时间的函数,可以针对日期和时间数据完成期望的功能。常用的日期和时间函数及其功能见表 2-10。

表 2-10　常用的日期和时间函数及其功能

日期和时间函数	功　　能
datepart(<日期分量>,<日期>)	以整数返回指定<日期>的指定<日期分量>
datename(<日期分量>,<日期>)	以字符串返回指定<日期>的指定<日期分量>
year(<日期>)	等价于 datepart(yy, <日期>)
month(<日期>)	等价于 datepart(mm, <日期>)
day(<日期>)	等价于 datepart(dd, <日期>)
dateadd(<日期分量>,<整数>,<日期>)	在指定<日期>的指定<日期分量>上和<整数>相加
datediff(<日期分量>,<开始日期>,<终止日期>)	按照指定<日期分量>用<开始日期>减<终止日期>
getdate( )	返回 SQL Server 所在机器的日期和时间

在 SQL 编辑器中输入下面语句,得到 getdate()函数的返回结果:2017-08-01 09:22:10.140。

```
SELECT getdate()
```

getdate()函数返回当前系统的日期和时间。可使用 datename()函数进一步把其中的年、月、日、时、分、秒等分量单独提取出来。日期分量及其含义见表 2-11。

表 2-11　日期分量及其含义

日期分量	缩　写	含　义
year	yy，yyyy	年(1900—2079)
quarter	qq，q	季(1～4)
month	mm，m	月(1～12)
dayofyear	dy，y	1～365
day	dd，d	1～31
week	wk，ww	周(1～52)
hour	hh	小时(1～24)
minute	mi，n	分钟(1～60)
second	ss，s	秒(1～60)
millisecond	ms	毫秒(1～1000)

year()、month()和 day()函数分别等价于 datepart(yyyy，＜日期＞)、datepart(mm，＜日期＞)和 datepart(dd，＜日期＞)。下面通过几个例子说明有关日期时间函数的用法。

例 47　设计 SQL 语句，从凭证库表中查询 2013 年的凭证。

```
SELECT *
FROM 凭证库
WHERE datepart(yyyy,凭证日期)=2013
```

或者使用等价的查询：

```
SELECT *
FROM 凭证库
WHERE year(凭证日期)=2013
```

例 48　设计 SQL 语句，从凭证库表中查询 1 月份的凭证。

```
SELECT *
FROM 凭证库
WHERE datepart(MM,凭证日期)=1
```

或者使用等价的查询：

```
SELECT *
FROM 凭证库
WHERE month(凭证日期)=1
```

例 49　设计 SQL 语句，从凭证库表中查询每月 1 日的凭证。

```
SELECT *
FROM 凭证库
```

```
WHERE datepart(dd,凭证日期) = 1
```

或者使用等价的查询：

```
SELECT *
FROM 凭证库
WHERE day(凭证日期) = 1
```

例 50 设计 SQL 语句,从凭证库表中查询 2013 年 4 月 1 日的凭证。

```
SELECT *
FROM 凭证库
WHERE year(凭证日期)= 2013 and month(凭证日期)= 4 and day(凭证日期) = 1
```

例 51 设计 SQL 语句,从凭证库表中查询 2013 年 4 月或 9 月的凭证。

```
SELECT *
FROM 凭证库
WHERE year(凭证日期)= 2013 and (month(凭证日期)= 4 or month(凭证日期)= 9)
```

例 52 查询当前日期后第 9 日的日期。

```
SELECT dateadd(dd,9,getdate()) AS '第 9 日'
```

例 53 假定某人的生日是 1971 年 2 月 20 日,计算其今年的年龄。

```
SELECT datediff(yy, '1971-02-20', getdate()) AS 年龄
```

2.6.3 数学函数

数学函数(如取绝对值函数 abs()等)完成与数学有关的计算。常见的数学函数及其功能见表 2-12。

表 2-12 常见数学函数及其功能

数学函数		功能
函数样例	返回值	
abs(－1.0)	1.0	绝对值
ceiling($123.45)	124	大于或等于所给数字表达式的最小整数
exp(2.1)	8.1661699125676517	返回所给的 float 表达式的指数值
floor(123.45)	123	返回小于或等于所给数字表达式的最大整数
log(5.175643)	1.6439635826406203	返回给定 float 表达式的自然对数
log10(100)	2.0	返回给定 float 表达式的以 10 为底的对数
power(2, 3)	8	2^3
rand(17)	0.713890120753055	返回 0～1 的随机 float 值,参数为种子

续表

数学函数		功　能
函数样例	返回值	
round(123.4545, 2) round(150.75, 0)	123.4500 151.00	四舍五入，小数点后保留2位 四舍五入，小数点后保留0位
sign(－100) sign(100) sign(0)	－1 1 0	符号函数
square(5)	25.0	求平方
sqrt(4)	2.0	求平方根
pi()		返回值为π，即3.1415926535897936

2.6.4　字符串函数

字符串函数对字符串输入值执行操作，返回字符串或数字值。表2-13列举了字符串函数及其功能。表中使用“□”表示空格字符。

表2-13　字符串函数及其功能

字符串函数		功　能
函数样例	返回值	
left('法律法规',2)	法律	返回从字符串左边开始指定个数的字符
right('法律法规条例',2)	条例	从指定字符串右边开始数，截取2个字符
substring('法律法规条例', 2, 3)	律法规	截取从指定字符串第2个字符开始的3个字符
len('法律法规') len('□□□□法律法规□□□□')	4 8	返回给定字符串表达式的字符(而不是字节)个数，其中不包含尾随空格
ltrim('□□□□法律法规')	法律法规	删除指定字符串中的起始空格
rtrim('□□□□法律法规□□□□')	□□□□法律法规	删除指定字符串中的尾随空格
replace('法律法规','法规','条文')	法律条文	使用'条文'替换'法律法规'中的'法规'
replicate('法律',3)	法律法律法律	重复指定字符串'法律'3次
reverse('法律法规')	法律法规	反转指定的字符串
charindex('法规','法律法规条例')	3	返回字符串中某个指定的子串出现的开始位置
stuff()		用另一子串填充字符串指定位置长度的子串
str(123.45, 6, 1)	□123.5	把数值123.45转换成字符，共6个字符位置(含小数点位置)，小数点后1个位置，四舍五入

续表

字符串函数		功　能
函数样例	返回值	
ascii('Abc')	65	返回字符表达式最左端字符的 ASCII 码值
char(65)	A	用于将 ASCII 码转换为字符
lower('ABc')	abc	把字符串全部转换为小写
upper('ABc')	ABC	把字符串全部转换为大写

下面通过几个例子说明字符串函数在查询设计中的应用。

例 54　设计 SQL 语句，从凭证库表中查询"摘要"的第二个字是"费"的行。

```
SELECT * FROM 凭证库 WHERE substring(摘要,2,1) = '费'
```

例 55　设计 SQL 语句，从凭证库表中查询"科目名称"的前 2 个字是"现金"的行。

```
SELECT *
FROM 凭证库
WHERE substring(科目名称,1,2) = '现金'
```

例 56　将数值 123.45 转换为长度为 6 的字符串，小数部分四舍五入保留一位。

```
SELECT str(123.45, 6, 1)
```

结果为：123.5

这里的参数 6 表示字符串的总长度，总长度包括小数点、符号、数字或空格，默认值为 10。参数 1 表示小数点右边的位数。

例 57　设计 SQL 语句，从凭证库表中查询"摘要"的第 1 个字是"费"的行。

```
SELECT * FROM 凭证库 WHERE left(摘要,1) = '费'
```

这里，left（摘要，1）返回从"摘要"左边开始的 1 个字符。

例 58　设计 SQL 语句，从凭证库表中查询"科目编码"为 101 的行及其下级科目的行。

```
SELECT * FROM 凭证库 WHERE left(科目编码,3)= '101'
```

例 59　设计 SQL 语句，从凭证库表中查询"科目编码"以"5"开头的行。

```
SELECT * FROM 凭证库 WHERE left(科目编码,1)= '5'
```

例 60　设计 SQL 语句，从凭证库表中查询有一级科目（一级科目为 3 位）的行。

```
SELECT * FROM 凭证库 WHERE len(科目编码)=3
```

例 61　设计 SQL 语句，从凭证库表中查询有二级科目（二级科目为 5 位）的行。

```
SELECT * FROM 凭证库 WHERE len(科目编码)=5
```

例 62　设计 SQL 语句，从凭证库表中查询一级科目和二级科目的行。

```
SELECT * FROM 凭证库 WHERE len(科目编码)<=5
```

例 63 设计 SQL 语句，从凭证库表中查询摘要为“招待费”的行。

```
SELECT * FROM 凭证库 WHERE rtrim(摘要)='招待费'
```

2.6.5 系统函数

系统函数完成一些通用的功能，如数据类型转换等。数据类型转换函数有下列两个。

```
CAST(<表达式>AS <数据类型>)把<表达式>转换为<数据类型>
CONVERT (<数据类型>[(长度)], <表达式>)把<表达式>转换为<数据类型>
```

下面的两个例子说明了类型转换函数在查询设计中的应用。

例 64 设计 SQL 语句，从凭证库表中查询“借方金额”中含有小数点的行。

```
SELECT 科目名称, 借方金额
FROM 凭证库
WHERE CAST(借方金额 AS char(20)) LIKE '%.%'
```

等价的查询语也可以是：

```
SELECT 科目名称, 借方金额
FROM 凭证库
WHERE CONVERT (char(20), 借方金额) LIKE '%.%'
```

例 65 设计 SQL 语句，从凭证库表中查询凭证日期为 2013 年的行。

```
SELECT 科目编码,科目名称,摘要,凭证日期,借方金额,贷方金额
FROM 凭证库
WHERE cast(凭证日期 as CHAR(19)) like '%2013%'
```

系统函数 ISNULL（＜被替换的表达式＞，＜替换表达式＞）的意思是：如果＜被替换的表达式＞为 NULL，则使用＜替换表达式＞将其替换；如果＜被替换的表达式＞不为 NULL，就返回该表达式的值。如果数据中可能包含空值，但不想在查询结果中出现空值，可以创建查询将空值转换成其他值。下面的示例显示了会计科目表中的所有科目，如果某科目的上级科目编码为 NULL，则显示“无”。

```
SELECT 科目编码,科目名称, ISNULL(上级科目编码,'无')
FROM 会计科目表
```

2.7 基于单表的查询技术

单表查询即从单个表中查询满足条件的行。满足什么样的条件，需要使用 WHERE 子句指定；如果需要把查询结果保存在表中，则使用 INTO 子句；如果需要小计，不是仅仅一个总计，则使用 GROUP BY 子句进行分组；如果需要对分组的结果进一步筛选，则使用 HAVING 子句；如果想对结果集排序，则使用 ORDER BY 子句。SELECT 语句的主要子句如下。

```
SELECT <项目列表>
[ INTO <新表> ]
FROM <表>
[ WHERE <检索条件> ]
[ GROUP BY <分组表达式> ]
[ HAVING <检索分组结果的条件表达式> ]
[ ORDER BY [ALL] <排序表达式> [ ASC | DESC] ]
```

下面分别说明这几个子句的使用方法。

2.7.1 WHERE 子句

使用 SELECT 和 FROM 仅可以从表中检索所有行。例如：

```
SELECT * FROM 凭证库
```

就从凭证库表中查询所有行。若要查找凭证库表中所有借方金额不低于 10000 元的行，则要使用 WHERE 子句指定检索条件。

```
SELECT * FROM 凭证库 WHERE 借方金额>=10000
```

WHERE 子句中的条件可以包含以下运算符。

1）比较运算符（如 =、<>、<和>等）

例如，从凭证库表中查询借方金额大于 10000 元的行。

```
SELECT * FROM 凭证库 WHERE 借方金额>10000
```

2）范围（BETWEEN…AND 和 NOT BETWEEN…AND）

例如，从凭证库表中查询 2013 年 12 月 1 日至 10 日期间的行。

```
SELECT * FROM 凭证库 WHERE 凭证日期 BETWEEN '2013-12-01' AND '2013-12-10'
```

3）一组值（IN 和 NOT IN）

例如，从凭证库表中查询凭证号为 10、11、12、13 和 14 的行。

```
SELECT * FROM 凭证库 WHERE 凭证号 IN ('10','11','12','13','14')
```

再如，从凭证库表中查询“科目名称”为银行存款、现金和应交营业税的行。

```
SELECT *
FROM 凭证库
WHERE 科目名称 IN('银行存款','现金','应交营业税')
```

4）模式匹配（LIKE 和 NOT LIKE）

例如，从凭证库表中查询“科目名称”不是以“现金”结尾的行。

```
SELECT * FROM 凭证库 WHERE 科目名称 NOT LIKE '%现金'
```

5）空值（IS NULL 和 IS NOT NULL）

例如，从会计科目表中查询所有的一级科目（无上级科目编码）。

```
SELECT * FROM 会计科目表 WHERE 上级科目编码 IS NULL
```

6）上述条件的组合（AND、OR、NOT）

例如，从凭证库表中查询科目代码以 101 开头且借方金额不小于 10000 元的行的摘要。

```
SELECT 摘要 FROM 凭证库 WHERE 科目编码 LIKE '101%' AND 借方金额>=10000
```

2.7.2 ORDER BY 子句

如果不使用 ORDER BY 子句，查询返回的结果顺序就是行录入时的顺序。如果希望以某种顺序返回查询结果，如按照金额大小、日期等排序，这时就要用到 ORDER BY 子句。

排序可以是升序（ASC）的，也可以是降序（DESC）的。如果没有指定是升序，还是降序，默认为 ASC。NULL 作为最小值处理。

下面通过几个例子说明 ORDER BY 子句的用法。

例 66 设计 SQL 语句，从凭证库表中查询所有凭证，查询结果按照借方金额由小到大排列。

```
SELECT * FROM 凭证库 ORDER BY 借方金额
```

注意，ORDER 和 BY 必须同时使用，默认是升序排序（ASC）。下面的查询语句与上面的查询效果相同。

```
SELECT * FROM 凭证库 ORDER BY 借方金额 ASC
```

如果要降序排序，则使用 DESC。

```
SELECT * FROM 凭证库 ORDER BY 借方金额 DESC
```

查询结果以升序排列或降序排列，并不意味着数据库表中的行也重新排序了。

以上例子仅按照一个列排序，也可以按照多个列排序。例如：

```
SELECT * FROM 凭证库 ORDER BY 借方金额,科目编码
```

该查询结果首先按照“借方金额”排序，对于“借方金额”相同的行，再按照“科目编码”排序。

如果按照多个列排序，则列的顺序不同，排序结果也不同。

```
SELECT * FROM 凭证库 ORDER BY 科目编码, 借方金额
```

上例中都是科目编码和借方金额的升序排序，如果要按照科目编码升序排序，相同科目编码的行按照借方金额降序排序，则使用如下语句。

```
SELECT * FROM 凭证库 ORDER BY 科目编码,借方金额 DESC
```

使用 TOP 关键字可以只显示查询结果集前面或后面的少数几个行。TOP 关键字通常与排序子句结合使用。其语法格式为

```
TOP n [percent] [WITH TIES]
```

其中，n为非负数。TOP n表示返回查询结果集的前n行；TOP n percent表示返回查询结果集的前n%行；WITH TIES表示如果最后一行还有与之在排序字段上取值相同的行，则仍然将其放到结果集中(这可能使得查询结果的总行数超过n)。

例67 设计SQL语句，从药品表中查询最贵的10种药品的编号和单价。

```
SELECT TOP 10 药品编号, 单价 FROM 药品 ORDER BY 单价 DESC
```

例68 从门诊收费明细表中查询"单价"最高的前4行。

```
SELECT TOP 10 门诊流水号, 单价 FROM 门诊收费明细 ORDER BY 单价 DESC
```

结果集为

```
门诊流水号      单价
--------------------
25587010014    1600.0000
25587010018    1600.0000
25587010020    1600.0000
25549010075    840.0000
```

如果使用WITH TIES：

```
SELECT TOP 4 WITH TIES 门诊流水号, 单价 FROM  门诊收费明细
ORDER BY 单价 DESC
```

则结果集变为

```
门诊流水号      单价
--------------------
25587010014    1600.0000
25587010018    1600.0000
25587010020    1600.0000
25583010010    840.0000
25583010010    840.0000
```

可以看到，第4行的单价值为"840"，由于表中还有一行的单价为"840"，因此"WITH TIES"就把该行显示出来了。

2.7.3 GROUP BY子句

假设已有"库存表"，表中有品名、颜色和数量3列，数据如下。

品　名	颜　色	数　量	品　名	颜　色	数　量
钢笔	蓝色	124	铅笔	红色	210
钢笔	红色	223	铅笔	红色	100
钢笔	蓝色	101	铅笔	蓝色	100

下面通过几个例子说明如何对"库存"表进行分组查询。

例 69　查询库存表中有几个“品名”。

```
SELECT 品名 FROM 库存 GROUP BY 品名
```

结果集为

```
品名
--------
钢笔
铅笔
```

例 70　按照“品名”小计其数量。

```
SELECT 品名, sum(数量)AS 小计 FROM库存 GROUP BY 品名
```

结果集为

```
品名        小计
--------------------
钢笔        448
铅笔        410
```

其中，品名“钢笔”的小计为 448，这就是“库存”表中品名为“钢笔”的 3 个数量 124、223、101 之和。

GROUP BY 关键字后面跟着分组列。GROUP BY 子句决定结果集中的行；每个结果集行都对应一个分组。

如果使用了 GROUP BY 子句，SELECT 子句中出现的表达式要么是分组列，要么是聚合函数。下面的查询语句中由于出现了非分组列“科目名称”，因此会出错。

```
SELECT 科目编码, 科目名称, sum(借方金额) AS 小计
FROM 凭证库
GROUP BY 科目编码
```

如果 SELECT 语句中既有 WHERE 子句，也有 GROUP BY 子句，则先根据 WHERE 子句执行筛选，对于满足条件的记录，再根据科目编码进行分组。例如：

```
SELECT 科目编码,sum(借方金额) AS 小计
FROM 凭证库
WHERE 借方金额>10000
GROUP BY 科目编码
```

当 SELECT 语句中有多个分组列时，则按照分组列的顺序依次进行分组。例如，查询语句：

```
SELECT 科目编码,会计月份,sum(借方金额) AS 小计
FROM 凭证库
WHERE 借方金额>10000
GROUP BY 科目编码,会计月份
```

中有两个分组列“科目编码”和“会计月份”，那么先按照“科目编码”分组，对于每个分组，再根据会计月份进行分组，如图 2-4 所示。

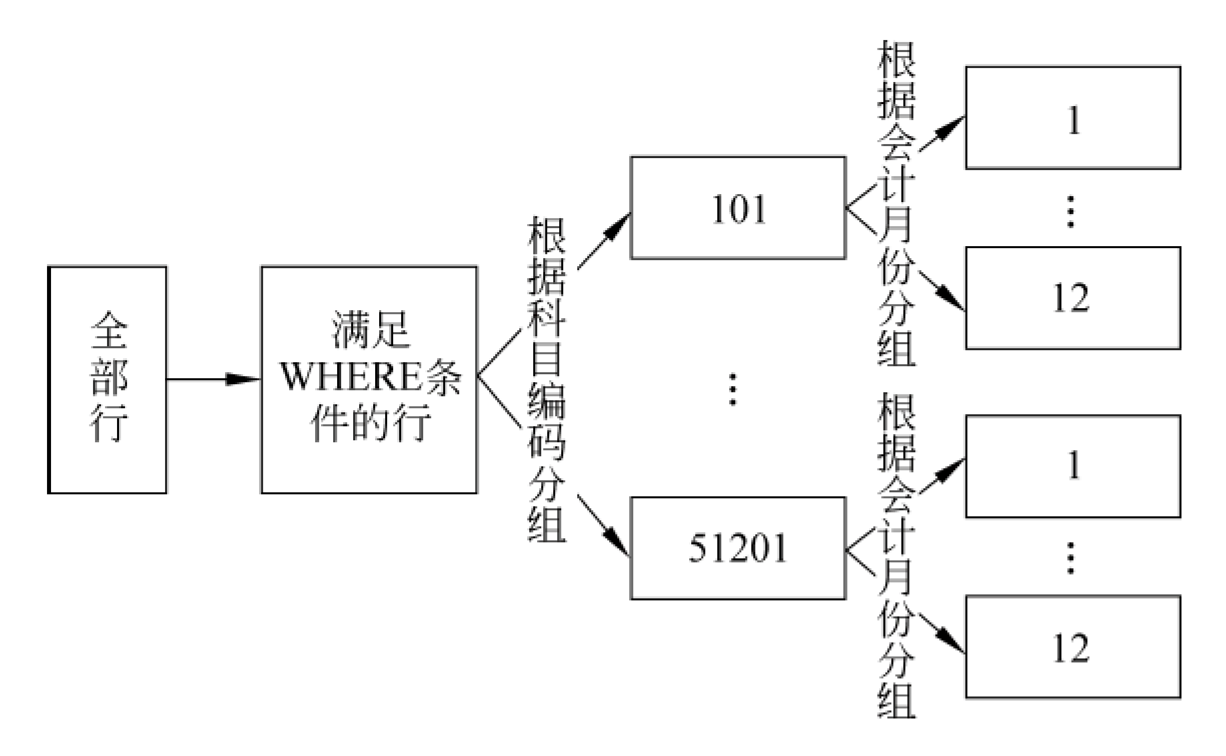


图 2-4　筛选、分组执行过程

在 SELECT 子句中可以同时使用多个聚合函数。例如，下面的查询使用了 sum()和 count()两个聚合函数。

```
SELECT 科目编码, sum(借方金额) AS 小计, count(借方金额) AS 行数
FROM 凭证库
WHERE 借方金额 >10000
GROUP BY 科目编码
```

对于分组后的查询结果集，还可以使用 ORDER BY 子句排序。例如：

```
SELECT 科目编码,sum(借方金额) AS 小计,count(借方金额) AS 行数
FROM 凭证库
WHERE 借方金额>10000
GROUP BY 科目编码
ORDER BY 科目编码
```

如果分组列包含一个空值，那么该行将成为结果中的一个分组；如果分组列包含多个空值，那么这些空值将放在一个分组中。假设“库存”表中的数据如图 2-5 所示。

id	品名	颜色	数量
1	钢笔	蓝色	124
2	钢笔	红色	223
3	钢笔	蓝色	101
4	铅笔	红色	210
5	铅笔	红色	100
6	铅笔	蓝色	100
7	NULL	蓝色	150
8	NULL	红色	150

图 2-5　含有空值的库存表

执行下面的 SQL 语句：

```
SELECT 品名, sum(数量) FROM 库存 GROUP BY 品名
```

查询结果集为

```
品名      (无列名)
NULL      300
钢笔      448
铅笔      410
```

其中，所有含有空值的行归到了一个分组中。

2.7.4 HAVING 子句

HAVING 子句用于对分组进行过滤，它的功能类似于 WHERE 子句，但它用于组，而不是单个行。WHERE 子句的检索条件在进行分组之前应用，而 HAVING 检索条件在进行分组后应用。HAVING 需要与 GROUP BY 子句一起使用，不能单独出现。

例 71 查找最大借方金额大于 10000 元的科目的编码和最大借方金额。

```
SELECT 科目编码, max(借方金额) AS 最大借方金额
FROM 凭证库
GROUP BY 科目编码
HAVING max(借方金额) > 10000
ORDER BY  max(借方金额)
```

注意，HAVING 后面的条件应是对聚合特性的约束；HAVING 子句中的聚合函数未必一定出现在 SELECT 子句中。例如，查找最大借方金额大于 10000 元，而且借方金额的个数大于 1 个的那些行，只显示科目编码和最大借方金额列。

```
SELECT 科目编码, max(借方金额) AS 最大借方金额
FROM 凭证库
GROUP BY 科目编码
HAVING max(借方金额) > 10000 AND count(借方金额)>1
```

例 72 组合使用 GROUP BY、HAVING 和 WHERE。将“凭证库”表中 2013 年 12 月 1 日和 2013 年 12 月 10 日之间的行按照科目编码分组计算借方金额小计，要求结果集中的借方金额小计大于 50000 元；并按照科目编码升序排序。

```
SELECT 科目编码, sum(借方金额) AS 小计
FROM 凭证库
WHERE 凭证日期 between '2013-12-01' and '2013-12-10'
GROUP BY 科目编码
HAVING  sum(借方金额) > 50000
ORDER BY 科目编码
```

WHERE、GROUP BY 和 HAVING 子句的作用和执行顺序如下：WHERE 子句用来筛选 FROM 子句指定的数据源；GROUP BY 子句用来对 WHERE 子句产生的结果进

行分组;HAVING 子句用来对分组后的结果再进行筛选,如图 2-6 所示。

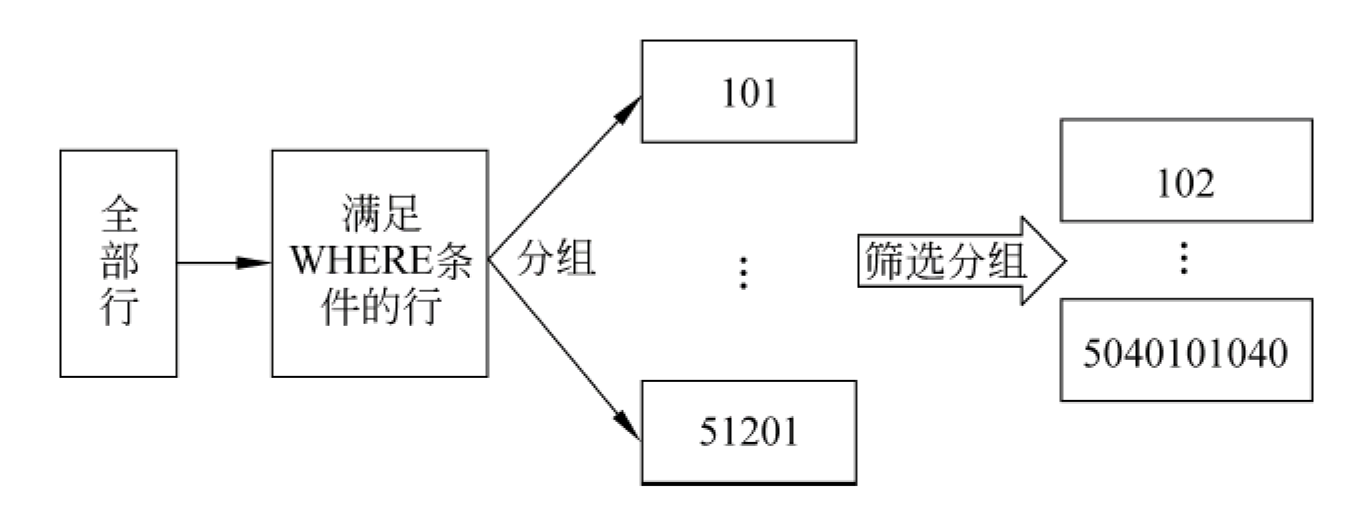


图 2-6 WHERE、GROUP BY 和 HAVING 子句的作用和执行顺序

2.7.5 持久化查询结果

T-SQL 允许把 SELECT 语句的结果集存放到新表中。创建新表使用 INTO 子句。新创建的数据表的属性列由 SELECT 语句的目标列表达式确定,属性列的列名、数据类型以及在表中的顺序都与 SELECT 语句的目标列表达式相同。新表的行数据也来自 SELECT 语句的查询结果。例如,使用 INTO 子句创建一个新表“药品代码名称对照”,存放药品表中的药品编号和名称两列。

```
SELECT 药品编号, 名称
INTO  药品代码名称对照
FROM 药品
```

通常情况下,INTO 子句的主要作用是创建一个临时表。因为使用 INTO 子句创建的表并不保留原表中已定义好的各种约束,因此一般不使用 INTO 子句创建永久表。创建临时表要求表名的前面有“#”前缀。例如:

```
SELECT 部门号 AS 部门, COUNT(*) AS 收费次数
INTO #部门收费次数对照
FROM 门诊收费
GROUP BY 部门号
```

临时表被创建在系统数据库 tempdb 中。

2.8 多表查询技术

在关系数据库中,为了避免冗余,数据通常分布在不同的表中。为了从两个或多个具有逻辑关系的表中查询,就要使用连接运算并指定连接条件。连接运算的语法为

```
FROM <表 1> <连接运算> <表 2> [ON <连接条件>]
```

其中,＜表 1＞、＜表 2＞指出参与连接操作的表名。连接可以对同一个表操作,也可以对多表操作。对同一个表操作的连接又称作自连接。

连接运算有 3 种类型:交叉连接、内连接和外连接。交叉连接(CROSS JOIN)返回参与连接运算的两个表中所有数据行的笛卡儿积,结果集合中的行数等于第一个表中的数

据行数乘以第二个表中的数据行数。

内连接(INNER JOIN)使用比较运算符进行表间列值的比较，返回参与连接的两个表中与连接条件相匹配的行。

外连接分为左外连接(LEFT JOIN)、右外连接(RIGHT JOIN)和全外连接(FULL JOIN)3 种。与内连接不同的是，外连接不只列出与连接条件相匹配的行，还列出左表(左外连接时)、右表(右外连接时)或两个表(全外连接时)中所有符合 WHERE 子句的行。

ON 子句指出连接条件。

2.8.1 交叉连接

交叉连接不需要与 ON 子句配套使用。例如，已知表 T1 和表 T2，T1 表有如下 3 行。

A	B	A	B
10	T1-1	30	T1-3
20	T1-2		

T2 表有如下 2 行。

A	B	C	A	B	C
10	T2-1	100	20	T2-2	200

交叉连接 T1 和 T2 的语句为

```
SELECT * FROM T1 CROSS JOIN T2
```

结果集为

A	B	A	B	C
10	T1-1	10	T1-1	100
10	T1-1	20	T2-2	200
20	T1-2	10	T2-1	100
20	T1-2	20	T2-2	200
30	T1-3	10	T2-1	100
30	T1-3	20	T2-2	200

T1 中有 3 行，T2 中有 2 行，两个表的交叉连接产生 3×2= 6 行。可见，交叉连接产生行的所有可能组合。在实践中并不经常使用交叉连接，但是交叉连接展示了所有连接的组合性质，并把参与连接的所有表的列整合在一起，包括其中的重复列。

2.8.2 内连接

内连接查询在交叉连接基础上进行筛选，找出满足连接条件的行。内连接可分等值

连接和不等值连接。等值连接是在连接条件中使用等于号(=)运算符比较连接条件涉及的列;不等值连接是在连接条件使用除了等于号(=)运算符以外的其他比较运算符比较连接条件涉及的列。这些运算符包括>、>=、<=、<、!>、!<和<>。

假设当前 T1 表中有如下 5 行。

A	B
10	T1-1
20	T1-2
30	T1-3
20	T1-4
20	T1-5

T2 表中有如下 3 行。

A	B	C
10	T2-1	100
20	T2-2	200
40	T2-3	400

把 T1 表中和 T2 表中在 A 列上取值相同的行连接在一起,即等值连接表 T1 和 T2 的查询语句为

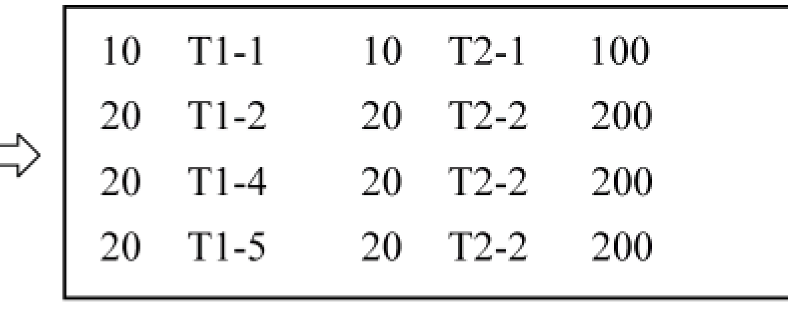
```
SELECT * FROM  T1  JOIN  T2  ON  T1.A=T2.A
```

结果集如图 2-7 所示。

A	B
10	T1-1
20	T1-2
30	T1-3
20	T1-4
20	T1-5

A	B	C
10	T2-1	100
20	T2-2	200
40	T2-3	400

⇨



10	T1-1	10	T2-1	100
20	T1-2	20	T2-2	200
20	T1-4	20	T2-2	200
20	T1-5	20	T2-2	200

图 2-7　等值连接

在列 A 上等值连接的结果使得 T1 中在列 A 上取值为 10 的行与 T2 中在列 A 上取值为 10 的行连接成结果集中的一行;T1 中在列 A 上取值为 20 的行与 T2 中在列 A 上取值为 20 的行连接成结果集中的一行。

上述内连接查询还可以写成如下形式。

```
SELECT * FROM  T1, T2 WHERE T1.A=T2.A
```

上述查询结果中包括了两个表的全部属性,如果查询结果只包括 T1.A、T1.B、T2.B,则相应的查询语句如下。

```
SELECT  T1.A, T1.B, T2.B
FROM  T1  JOIN  T2  ON  T1.A=T2.A
```

注意,由于在两个表中都有列 A、列 B,所以要在 SELECT 子句中通过<表名>.<列名>指明查询结果中的列来自哪个表。

再看一个例子。在门诊收费表中只有部门号,没有部门名称。如果希望在结果集中

包含每笔门诊收费的“部门名称”，则需要把门诊收费表及部门表连接起来。

```
SELECT *
FROM 门诊收费
SELECT *
FROM 门诊收费  JOIN 部门 ON 门诊收费.部门号 =部门.部门号
```

去掉不关心的列，并把部门表中的部门名称移到“门诊收费”表的部门号后面：

```
SELECT 门诊流水号, 门诊收费.部门号, 名称, 收费日期
FROM 门诊收费  JOIN 部门 ON 门诊收费.部门号 =部门.部门号
```

在上例中重复书写表名“门诊收费”很烦琐，为了简化书写，可在 FROM 子句中通过 AS 为表指定别名。

```
SELECT 门诊流水号,M.部门号,名称,收费日期
FROM 门诊收费 AS M  JOIN 部门 AS B ON M.部门号 =B.部门号
```

在这个例子中，为门诊收费表指定了别名 M；为部门表指定了别名 B。注意，一旦为表指定了别名，则在查询语句的其他所有用到表名的地方都要使用别名，而不能使用原表名。

如果只关心门诊流水号为 24820010001 的那条记录，则可以在上面的 SQL 语句基础上增加 WHERE 子句：

```
SELECT 门诊流水号,M.部门号,部门名称,收费日期
FROM 门诊收费 AS M  JOIN 部门 AS B ON M.部门号 =B.部门号
WHERE 门诊流水号 ='24820010001'
```

数据库引擎执行查询语句时，首先为被连接的表生成笛卡儿积；然后，从笛卡儿积中选择满足 ON 子句中连接条件的行；接下来，使用 WHERE 子句从中进一步进行筛选；如果有 GROUP BY 子句，则按照指定的分组列进行分组；如果有 HAVING 子句，则按照指定条件对分组结果进行筛选；最后，筛选出在 SELECT 子句中指定的列。如果有 ORDER BY 子句，则按照排序列进行排序。

下面两个例子是在两个表上做聚合查询。

例 73 设计查询语句，按照部门名称分组汇总门诊收费次数。

```
SELECT 名称,count(M.门诊流水号)
FROM 门诊收费 AS M  JOIN 部门 AS B ON M.部门号 =B.部门号
GROUP BY 名称
```

例 74 设计查询语句，按照部门名称分组汇总 2014 年 1 月份的门诊收费次数。

```
SELECT 名称,count(M.门诊流水号)
FROM 门诊收费 AS M  JOIN 部门 AS B ON M.部门号 =B.部门号
WHERE M.收费日期 BETWEEN '2014/01/01' AND '2014/01/31'
GROUP BY 名称
```

连接运输也可以涉及 3 个或更多的表。

```
SELECT M.门诊流水号,M.项目编号, N.部门号, B.名称
FROM 门诊收费明细 AS M
JOIN 门诊收费 AS N ON M.门诊流水号 =N.门诊流水号
JOIN 部门 AS B ON N.部门号 =B.部门号
```

连接把两个或者两个以上表中的列整合到结果集中。对连接后的结果集还能够应用筛选、分组、排序等操作。

例如,假设当前“药品”表中有如下 5 行。

药 品 编 号	药 品 名 称	单价/元
A1-1	胃复安片	10
A1-2	维生素 B1 片	10
A1-3	胃舒平	30
B1-4		20
B1-5		20

“收费明细”表中有如下 3 行。

项目编号	单价/元	数 量	项目编号	单价/元	数 量
A1-1	10	100	A1-3	30	400
A1-2	10	200			

下面的语句用于查询收费明细表中涉及哪些药品。

```
SELECT *
FROM 收费明细 JOIN 药品 ON 收费明细.项目编号 =药品.药品编号
```

结果集为

项目编号	单价/元	数量	药品编号	药品名称	单价/元
A1-1	10	100	A1-1	胃复安片	10
A1-2	10	200	A1-2	维生素 B1 片	10
A1-3	30	400	A1-3	胃舒平	30

语句:

```
SELECT DISTINCT 药品名称
FROM 收费明细   JOIN 药品 ON 收费明细.项目编号 =药品.药品编号
```

的查询结果为

```
药品名称
----------
```

```
维生素 B1 片
胃复安片
胃舒平
```

连接两个表时，选择什么样的列放在连接条件中进行连接呢？一般来说，应选择具有“引用”关系的列。例如，收费明细表中的“项目编号”列引用了药品表中的“药品编号”列，那么这两列就可作为连接条件中的列。被引用的列一般能够唯一标识其所在行。例如，在药品表中，“药品编号”能够唯一标识其所在行，即如果药品编号不同，其所在行也不同；如果所在行不同，其在“药品编号”列上的取值也不同。

由于收费明细表中的“单价”和药品表中的“单价”不具有引用关系，所以不能依据“单价”进行连接。

2.8.3 自连接

自连接是一种特殊的连接，被连接的两个表在物理上是一个表，但在逻辑上是两个表。

例如，从“会计科目表”查询所有会计科目的科目编码、科目名称、上级科目编码、上级科目名称。

```
SELECT M.科目编码, M.科目名称, M.上级科目编码, N.科目名称 AS 上级科目名称
FROM 会计科目表 AS M  JOIN 会计科目表 AS N on M.上级科目编码=N.科目编码
ORDER BY M.科目编码
```

注意：如果连接列在某行上出现 NULL 值，即使两个表的连接列都是 NULL，连接结果也不会存在 NULL。因为比较表达式一侧是 NULL 或者 NULL = NULL 的时候，计算结果都是 Unknown。因此，该查询不包括顶级科目的信息。

2.8.4 外连接

内连接时，返回到查询结果集合的仅是符合连接条件和查询条件（WHERE 条件或 HAVING 条件）的行。而采用外连接时，返回到查询结果集合中的不仅包含内连接产生的行，还包括左表（左外连接时）、右表（右外连接时）或两个连接表（全外连接）中的符合查询条件的数据行。对于这些行，在另外一个表中没有一行能够与它满足连接条件，因此这些行不会出现在内连接中。

例如，假设当前 T1 表中有如下 5 行。T2 表中有如下 3 行。

T1：

A	B
10	T1-1
20	T1-2
30	T1-3
20	T1-4
20	T1-5

T2：

A	B	C
10	T2-1	100
20	T2-2	200
40	T2-3	400

如果希望在结果集中包含表T1的所有行，若表T2中没有与之对应的行，则在结果集中置为空，如图2-8所示。

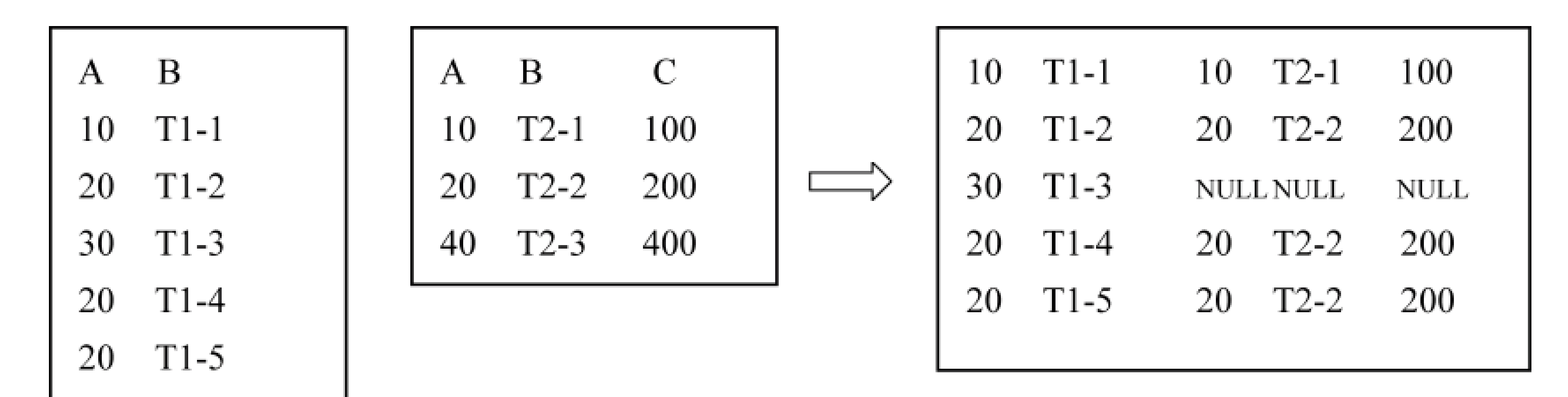


图2-8 外连接

这种情况下使用左连接：

```
SELECT *
FROM  T1  LEFT JOIN  T2  ON  T1.A = T2.A
```

也可以通过右外连接实现。

```
SELECT *
FROM  T2  RIGHT JOIN  T1  ON  T1.A = T2.A
```

如果希望表T1的所有行在结果集中都出现，若表T2中没有与之对应的行，则在结果集中置为空；同时要求表T2的所有行在结果集中都出现，若表T1中没有与之对应的行，则在结果集中置为空。那么，查询语句为

```
SELECT *
FROM  T1 FULL  JOIN  T2ON  T1.A = T2.A
```

结果集如下。

```
A     B          A     B          C
10    T1- 1      10    T2- 1      100
20    T1- 2      20    T2- 2      200
30    T1- 3      NULL  NULL       NULL
20    T1- 4      20    T2- 2      200
20    T1- 5      20    T2- 2      200
NULL  NULL       40    T2- 3      400
```

如果连接条件中的列有空值，则这些空值互相不匹配。例如，下面的两个表T1和T2中，每个表在要参与连接的列中均包含NULL值。

T1：

A	B	A	B
10	T1-1	20	T1-4
NULL	T1-2	20	T1-5
30	T1-3		

T2：

A	B	C	A	B	C
10	T2-1	100	40	T2-3	400
20	T2-2	200	NULL	T2-4	600

将表 T1 的列 A 中的值与表 T2 的列 A 中的值进行比较连接，在包含 NULL 值的列上不能获得匹配结果。

```
SELECT * FROM T1 JOIN T2 ON T1.A = T2.A
```

结果集为

A	B	A	B	C
10	T1-1	10	T2-1	100
20	T1-4	20	T2-2	200
20	T1-5	20	T2-2	200

下面的语句对这两个表进行左连接：

```
SELECT *
FROM T1 LEFT JOIN T2 ON T1.A = T2.A
```

结果集为

A	B	A	B	C
10	T1-1	10	T2-1	100
NULL	T1-2	NULL	NULL	NULL
30	T1-3	NULL	NULL	NULL
20	T1-4	20	T2-2	200
20	T1-5	20	T2-2	200

下面的语句对这两个表进行右连接：

```
SELECT *
FROM T2 RIGHT JOIN T1 ON T1.A = T2.A
```

结果集为

A	B	C	A	B
10	T2-1	100	10	T1-1
NULL	NULL	NULL	NULL	T1-2

续表

A	B	C	A	B
NULL	NULL	NULL	30	T1-3
20	T2-2	200	20	T1-4
20	T2-2	200	20	T1-5

下面的语句对这两个表进行全外连接：

```
SELECT *
FROM T1   FULL JOIN   T2 ON   T1.A = T2.A
```

结果集为

A	B	A	B	C
10	T1-1	10	T2-1	100
NULL	T1-2	NULL	NULL	NULL
30	T1-3	NULL	NULL	NULL
20	T1-4	20	T2-2	200
20	T1-5	20	T2-2	200
NULL	NULL	40	T2-3	400
NULL	NULL	NULL	T2-4	600

T-SQL 对子句的处理顺序是：FROM、ON、JOIN、WHERE、GROUP BY、HAVING、SELECT、DISTINCT、ORDERBY 和 TOP。

最后介绍一个较为复杂的应用。科目代码通常采用层次编码，在审计实践中，往往需要根据某个具体的科目编码立刻看到其相关的各级科目名称。

例如，从被审单位得到的会计科目表如图 2-9 所示。

表"会计科目表"中的数据，位置是"东山县医院...

科目编码	科目名称	余额方向	科目类别名称	上级科目编码
3030304	售房款	-1	<NULL>	30303
3030305	维修基金	-1	<NULL>	30303
30601	事业结余	-1	<NULL>	306
30802	提取职工福利费	-1	<NULL>	308
30803	转入事业基金	-1	<NULL>	308
30804	未分配结余	-1	<NULL>	308
41201	通达中心	-1	<NULL>	412
41202	海立中心	-1	<NULL>	412
40501	事业收入	-1	<NULL>	405
41203	鲁上基地	-1	<NULL>	412
50402	项目支出	1	<NULL>	504
5040101	人员支出	1	<NULL>	50401
50401010205	电话补贴	1	<NULL>	504010102
50401010206	伙食补贴	1	<NULL>	504010102
504010103	奖金	1	<NULL>	5040101
504010104	社会保障缴费	1	<NULL>	5040101

图 2-9　会计科目表

如果希望以图 2-10 形式观察数据：

	科目编码	科目名称	一级科目名称	二级科目名称	三级科目名称	四级科目名
1	101	现金	现金	现金	现金	现金
2	102	银行存款	银行存款	银行存款	银行存款	银行存款
3	110	其他应收款	其他应收款	其他应收款	其他应收款	其他应收款
4	11001	预支差旅费	其他应收款	预支差旅费	预支差旅费	预支差旅费
5	11002	海渔业协会	其他应收款	海渔业协会	海渔业协会	海渔业协会
6	11003	海立中心	其他应收款	海立中心	海立中心	海立中心
7	11004	通达中心	其他应收款	通达中心	通达中心	通达中心

图 2-10　期望看到的会计科目表

那么，可应用连接在查询编辑器中执行下列查询。

```
/*
32222 编码
任意一科目所属一级科目是其编码的前 3 位
*/
SELECT A.科目编码,A.科目名称,
B.科目名称 AS 一级科目名称,
C.科目名称 AS 二级科目名称,
D.科目名称 AS 三级科目名称,
E.科目名称 AS 四级科目名称,
F.科目名称 AS 五级科目名称
INTO 会计科目一览表
FROM 会计科目表 AS A JOIN 会计科目表 AS B ON B.科目编码 =left(A.科目编码,3)
        JOIN 会计科目表 AS C ON C.科目编码 =left(A.科目编码,5)
        JOIN 会计科目表 AS D ON D.科目编码 =left(A.科目编码,7)
        JOIN 会计科目表 AS E ON E.科目编码 =left(A.科目编码,9)
        JOIN 会计科目表 AS F ON F.科目编码 =left(A.科目编码,11)
ORDER BY A.科目编码
UPDATE 会计科目一览表
SET 五级科目名称 =NULL
WHERE len(科目编码) <11
UPDATE 会计科目一览表
SET 四级科目名称 =NULL
WHERE len(科目编码) <9
UPDATE 会计科目一览表
SET 三级科目名称 =NULL
WHERE len(科目编码) <7
UPDATE 会计科目一览表
SET 二级科目名称 =NULL
WHERE len(科目编码) <5
```

2.9　子查询技术

子查询是一个嵌套在 SELECT 语句中的 SELECT 语句。应用子查询可以把若干个查询语句整合到一个查询语句中。

2.9.1 使用返回单个值的子查询

如果一个 SELECT 语句的结果集中只有一个值,就可以在另外一个查询的关系表达式中嵌入该 SELECT 语句。下面通过几个例子说明。

例 75 查询脑血管科的门诊收费情况。

门诊收费表中只包括部门号,没有部门名称;而题目只给出了部门名称。所以,解决该问题可以分两步。

(1) 首先,从部门表中根据部门名称查找该部门的部门号。

```
SELECT 部门号
FROM 部门
WHERE 名称 = '脑血管科'
```

(2) 然后,在门诊收费表中根据部门号查找该部门的门诊收费记录。

```
SELECT *
FROM 门诊收费
WHERE 部门号 = '10201'
```

子查询技术可以将以上两个查询语句合二为一:

```
SELECT *
FROM 门诊收费
WHERE 部门号 = (SELECT 部门号
                FROM 部门
                WHERE 名称 = '脑血管科')
```

括号内的查询称为"子查询";括号外的查询称为"外部查询"。比较运算符(=、<>、>、>=、<、!>、!<或<=)都可被应用在子查询上,但要求子查询必须返回单个值(一行一列)。如果这样的子查询返回多个值,SQL Server 将显示错误信息。所以,在设计查询之前,必须对数据的组织和问题的本质非常熟悉,以确保该子查询只返回一个值。

例 76 从门诊收费明细表中查询超出平均金额的收费记录。

```
SELECT *
FROM 门诊收费明细
WHERE 单价 * 数量> (SELECT avg(单价 * 数量)
          FROM 门诊收费明细)
```

括号中的 SELECT 语句(子查询)使用聚合函数 avg,结果仍然是单个值。

例 77 查询与医生"刘江"在同一部门的医生。

```
SELECT 姓名
FROM 医生
WHERE 部门号 = (SELECT 部门号
                  FROM 医生
                  WHERE 姓名 = '刘江')
```

该查询首先执行子查询，根据姓名“刘江”从医生表中查询其所在的部门号，然后根据这个部门号从医生表中查询该部门所有医生的姓名。

2.9.2 使用返回多个值的子查询

如果子查询结果含有多个值，即结果集中有一列多行，不能在子查询上使用比较运算符，而是使用谓词 IN。下面通过例子说明。

例 78 如果有两个医生姓名都是“刘江”，且可能在不同部门，那么要查找与医生“刘江”属于同一部门的医生，下面的方法更准确。

```
SELECT 姓名
FROM 医生
WHERE 部门号 IN (SELECT 部门号
                FROM 医生
                WHERE  姓名 ='刘江')
```

这个查询考虑了子查询可能返回多个值，使用谓词 IN 与多个值进行比较，只要与多个值中的一个相等，则谓词 IN 就返回“真”。

例 79 查询开出药品“A010254”的所有医生的医生号。

```
SELECT DISTINCT 医生号
FROM 门诊收费
WHERE 门诊流水号 IN (SELECT 门诊流水号
                    FROM  门诊收费明细
WHERE 项目编号='A010254')
```

该语句分两步执行：首先，子查询返回满足条件的门诊流水号；然后，这些值被代入外部查询中，在门诊收费表中查找相匹配的行。

例 80 查询名称中含“葡萄糖”字样的门诊收费明细中的行。

```
SELECT *
FROM 门诊收费明细
WHERE  项目编号 IN (SELECT 药品编号
                   FROM  药品
                   WHERE 名称 LIKE '%葡萄糖%')
```

例 81 查询名称中不含“葡萄糖”字样的门诊收费明细中的行。

```
SELECT *
FROM 门诊收费明细
WHERE  项目编号 NOT IN (SELECT 药品编号
                       FROM  药品
                       WHERE 名称 LIKE '%葡萄糖%')
```

2.9.3 应用子查询进行存在性测试

在子查询上应用 EXISTS 关键字就相当于进行存在性测试。EXISTS 判断子查询的

结果集是否为空。也就是说，如果子查询的结果集有1行或者多行，则EXISTS返回“真”；如果一行也没有，则EXISTS返回“假”。例如，如果希望查询哪些“部门”具有收费行为，则可以设计如下子查询。

```
SELECT *
FROM 部门
WHERE  EXISTS (SELECT *
               FROM  门诊收费
               WHERE 门诊收费.部门号 =部门.部门号)
```

执行过程：

（1）无条件执行外层查询，在其结果中取第一行作为当前行。

（2）将当前行中的“部门号”代入子查询的WHERE子句中，执行子查询。如果子查询的结果集中的行数大于或等于1，则EXISTS的计算结果为“存在”，返回TRUE；否则，EXISTS的计算结果为“不存在”，返回FALSE。

（3）取当前行的下一行作为当前行，重复执行步骤（2），直到全部行被处理完。

如果查询没有收费行为的部门，则设计如下子查询。

```
SELECT *
FROM 部门
WHERE NOT EXISTS (SELECT *
                  FROM  门诊收费
                  WHERE 门诊收费.部门号 =部门.部门号)
```

注意：EXISTS关键字前面没有列名、常量或其他表达式；EXISTS后的子查询的SELECT子句选择列表通常由星号（*）组成。这是由于只是测试是否存在符合子查询中指定条件的行，所以不必指定列名；子查询的执行次数由外部查询中表的行数决定。每次执行子查询的结果会随着外层查询当前行的变化而变化。

多数情况下，子查询存在一个等价的连接查询。例如，查询语句：

```
SELECT *
FROM 门诊收费
WHERE 部门号 = (SELECT 部门号
               FROM 部门
               WHERE 部门名称 = '脑血管科')
```

也可以通过内连接得到。

```
SELECT M.*
FROM 门诊收费 AS M  JOIN 部门 AS B ON M.部门号 =B.部门号
WHERE 部门名称 = '脑血管科'
```

再如，查询语句：

```
SELECT DISTINCT 医生号
FROM 门诊收费
```

```
WHERE 门诊流水号 IN ( SELECT 门诊流水号
                    FROM 门诊收费明细
                    WHERE 项目编号 = 'A010254')
ORDER BY 医生号
```

也可以通过内连接得到：

```
SELECT DISTINCT 医生号
FROM 门诊收费 AS M JOIN 门诊收费明细 AS N ON M.门诊流水号 =N.门诊流水号
WHERE  N.项目编号= 'A010254'
ORDER BY 医生号
```

但是，连接查询未必存在等价的子查询，子查询也未必存在等价的连接查询。如果查询结果的列涉及多个表，则一般使用连接查询。

2.10 合　　并

UNION 运算符将两个或多个 SELECT 语句的结果合并成一个结果集。使用 UNION 合并的结果集都必须具有相同的结构，即它们的列数必须相同，并且对应列的数据类型必须兼容。假设 T1 表和 T2 表如下。

T1：

A	B
10	T1-1
20	T1-2
30	T1-3
20	T1-4
20	T1-5

T2：

A	B	C
10	T2-1	100
20	T2-2	200
40	T2-3	400

如果希望把 T1 表的所有行和 T2 表的所有行合在一起，形成：

A	B
10	T1-1
20	T1-2
20	T1-4
20	T1-5
30	T1-3
10	T2-1
20	T2-2
40	T2-3

则使用 UNION 合并两个查询的结果集。

```
SELECT A,B
FROM T1
UNION
SELECT A,B
FROM T2
```

UNION 的结果集列名与 UNION 运算符中第一个 SELECT 语句的结果集中的列名相同。另一个 SELECT 语句的结果集列名将被忽略。

默认情况下,UNION 运算符从结果集中删除重复的行。如果使用 ALL 关键字,那么结果中将包含所有行并且不删除重复的行。

再如,从"会计科目"表中查询所有会计科目的科目编码、科目名称、上级科目编码、上级科目名称的 SQL 语句为

```
SELECT m.科目编码,m.科目名称,m.上级科目编码,n.科目名称 as 上级科目名称
FROM 会计科目表 as m  join 会计科目表 as n on m.上级科目编码=n.科目编码
ORDER BY m.科目编码
```

但是,查询结果中缺失了顶级会计科目。此时可以使用 UNION 把顶级会计科目合并到结果集中。

```
SELECT  m.科目编码,m.科目名称,m.上级科目编码,n.科目名称 as 上级科目名称
FROM 会计科目表 AS m  join 会计科目表 AS n on m.上级科目编码=n.科目编码
UNION
SELECT  科目编码,科目名称,'无','无'
FROM 会计科目表
WHERE 上级科目编码 is NULL
ORDER BY  m.科目编码
```

2.11 修改数据

修改表中的数据包括向表中插入一些行、改变一些行上的值或者删除一些行。

2.11.1 插入行

INSERT 语句实现插入一些行。INSERT 语句有两种形式:一种是使用 VALUES 子句,一次只能插入一行;另一种是插入 SELECT 子句子查询的结果,一次可插入多行。

使用 VALUES 子句插入一行的语法为

```
INSERT INTO <表名> [(<若干列名>)]
VALUES (<若干值>)
```

该命令将单条新行插入指定表中,列名与值根据位置对应。

例 82 设计 SQL 语句，向部门表新增部门“信息中心”，其部门号是 5，级别是 1，不是叶子节点。

```
INSERT INTO 部门 (部门号, 名称, 级别, 是否叶子)
VALUES ('5','信息中心',1,0)
```

该语句与下面的语句等价：

```
INSERT INTO 部门 VALUES ('5','信息中心',1,0)
```

再插入一部门“医学影像中心”：

```
INSERT INTO 部门 VALUES ('6','医学影像中心',1,0)
```

注意：列的个数与值的个数要相等，且顺序一致，否则会产生语法错误。如果 INTO 子句后没有指定任何列，则 VALUES 子句后面的值的个数必须与表中列的个数相等，且类型、顺序一致，否则会出语法错误或导致赋值不正确。

例 83 设计查询，向医生表新增医生，姓名为“陈淳”，医生号为 501，部门号为 5。

```
INSERT INTO 医生
VALUES('501','陈淳','5')
```

再插入一名医生“陈美”：

```
INSERT INTO 医生 (医生号,姓名)
VALUES('502','陈美')
```

INSERT 语句要满足表的完整性约束，否则执行将失败。常见的约束如下。

(1) 主键(Primary Key)约束：在表中，如果某列或者某些列上的值用来唯一地标识其所在行，那么该值称为主键值，该列称为主键列。主键值能够唯一确定表中的每一行，主键不能取空值。如果部门表中的“部门号”是主键，如果要插入的行的部门号在部门表中已经存在了，则插入操作将失败。

(2) 外键(Foreign Key)约束：在表 A 中的某列是对表 B 中的主键列的引用，那么该列称为表 A 的外键。例如，医生表的“部门号”是其外键，引用部门表的“部门号”。如果要插入一条医生的信息，则其所属部门号应该是部门表中的某个部门号，或者取空值，否则操作将失败。

(3) 唯一(UNIQUE)约束：有些属性或属性组虽然不是主键，但仍可以限制其取值唯一，对于这些属性，如果 INSERT 语句试图插入一个表中已经存在的值，将失败。具有 UNIQUE 约束的列中不能出现重复值，但允许出现一次空值。

(4) 非空(NOT NULL)约束：如果某字段可以接受空值，则 INSERT 语句中可以忽略该字段，否则必须赋值(例如，“部门”表中的“名称”列不能为空)。

(5) 默认(DEFAULT)约束：如果在插入操作中没有为具有非空约束的列提供值，则取该值。

(6) 检查(CHECK)约束：对输入列或整个表中的值设置检查条件，以限制输入值(例如，“部门”表中的“级别”列只能取 1、2 或 3)。

SELECT 语句可以作为子查询嵌套在 INSERT 语句中,用以插入批量行。其语句格式一般为

```
INSERT INTO <表名> [(<若干列名>)]
```

例如,查询每个部门门诊收费次数,并将部门号和门诊收费次数放入一张新表“部门收费”中。那么,先应用下面的语句创建新表。

```
CREATE TABLE 部门收费 (部门号 CHAR(20),收费次数 SMALLINT)
```

然后应用 INSERT INTO 把 SELECT 语句的查询结果集插入新表中。

```
INSERT INTO 部门收费 (部门号, 收费次数)
SELECT 部门号, count(门诊流水号)
FROM 门诊收费
GROUP BY 部门号
```

2.11.2 修改行

SQL 修改行的语句为 UPDATE。其一般语句格式为

```
UPDATE <表名>
SET <列名>=<表达式>[,<列名>=<表达式>][,…]
[WHERE <条件>]
```

当有 WHERE 子句时,修改满足指定条件(WHERE 子句)的行;当没有 WHERE 子句时,修改全部行。例如,把 501 号医生的姓名修改为“王丹”的语句为

```
UPDATE 医生
SET 姓名='王丹'
WHERE 医生号='501'
```

又如,修改 501 号医生的姓名为“王丹晨”、部门号为“10101”的语句为

```
UPDATE 医生
SET 姓名='王丹晨', 部门号='10101'
WHERE 医生号='501'
```

把所有药品单价上调 5%的语句为

```
UPDATE 药品
SET 单价=单价 * 1.05
```

注意:在 UPDATE 语句中对批量行的修改要慎重。实际应用中,这样的批量行往往数以万计,因此 WHERE 条件范围一定要设计好,否则一旦修改错了,要恢复就比较麻烦。一个稳妥的做法是:在使用 UPDATE 语句之前,先用 SELECT 语句将要修改的行找出来检查一遍,确定正确无误后再修改。

UPDATE 语句要满足表的完整性约束,否则执行将失败。常见的约束包括主键约束、外键约束、唯一性约束、默认约束、非空约束、检查约束等。

2.11.3 删除行

SQL 删除行的语句是 DELETE。DELETE 语句的一般格式为

```
DELETE  FROM <表名>  [WHERE <条件>]
```

当有 WHERE 子句时，删除满足指定条件（WHERE 子句）的行；当没有 WHERE 子句时，删除所有行（表将成为空表）。

例如，删除部门编号为 6 的部门：

```
DELETE
FROM 部门
WHERE 部门编号='6'
```

在 DELETE 语句中也可以应用子查询。例如，从门诊收费表中删除部门名称为“脑血管科”和“信息中心”的行：

```
DELETE
FROM 门诊收费
WHERE 部门号 IN (SELECT 部门号
                 FROM 部门
                 WHERE 名称='脑血管科' OR 名称='信息中心')
```

因为 SELECT 子查询得到的行一般不止一行，所以子查询前面的运算符不能为“＝”，应该用表示取值范围的“IN”。

清空部门收费表的语句为

```
DELETE
FROM 部门收费
```

删除所有行即清空表中的数据，因此不加 WHERE 子句。这类操作的执行应非常小心。

DELETE 语句要满足表的完整性约束，否则执行将失败。常见的约束主要是外键约束。例如，医生表的“部门号”是其外键，引用了部门表的“部门号”，如果要删除部门表中的某行，是否能够成功取决于是否满足该参照完整性所设置的删除策略。

另外，TRUNCATE TABLE 语句也可以清空表，该语句的执行速度比 DELETE 语句快。语法如下。

```
TRUNCATE TABLE <表名>
```

2.12 应用 DDL 管理表

管理表包括创建、修改、删除表，设置完整性约束规则等。数据定义语言（DDL）可用于对表的管理。

定义表就是定义表的名字、列的名字、数据类型以及列上的约束等。例如，定义部门表：

```
CREATE TABLE 部门
(
部门号   nvarchar (20)   PRIMARY KEY ,
名称 nvarchar(50)   NOT NULL   UNIQUE ,
类别 tinyint NOT NULL   CHECK (类别=1 or 类别=2 or 类别=3),
是否叶子 bit NOT NULL
)
```

其中,PRIMARY KEY 指定主键,主键可以建立在单个列上,也可以建立在多个列上。主键用以保证实体完整性,即要求每一个表中的主键字段都不能为空或者不能为重复的值,以起到行的唯一标识作用。对于每个表,只能创建一个 PRIMARY KEY 约束;NOT NULL 表示列中不允许空值;UNIQUE 表示该列不能出现重复值;CHECK (类别=1 or 类别=2 or 类别= 3) 是 CHECK 约束对输入到部门表中的类别列的值的限制。

以下是医生表的定义语句。

```
CREATE TABLE 医生
(
医生号 nvarchar(20)   PRIMARY KEY ,
姓名    nvarchar(20)   NOT NULL   DEFAULT '未知' ,
部门号 nvarchar(20)   NOT NULL   REFERENCES 部门 (部门号)
                               ON DELETE CASCADE
                               ON UPDATE CASCADE
)
```

“REFERENCES 部门(部门号)”定义医生表引用部门表的列“部门”,这是 FOREIGN KEY 约束,是为列中的数据提供引用完整性的约束。FOREIGN KEY 约束要求列中的每个值在被参照表中对应的被引用列中都存在。FOREIGN KEY 约束只能引用被参照表中 PRIMARY KEY 或 UNIQUE 约束的列。

ON DELETE CASCADE 在外键约束上设计“级联删除”策略:当在部门表中删除某个部门时,也要把医生表中相应部门的医生全部删除;ON UPDATE CASCADE 在外键约束上设置“级联更新”策略:当修改部门表中某个部门号时,也要修改医生表中相应部门的医生所属的部门号。

在创建 FOREIGN KEY 约束之前,被参照表(本例中为部门表)要先创建好,医生表称为参照表。

数据完整性指存储在数据库中的所有数据值均正确的状态。如果数据库中存储有不正确的数据值,则该数据库称为已丧失数据完整性。

强制数据完整性可确保数据库中的数据一致性。例如,如果在医生表中输入了医生号为 123 的医生,那么该数据库不应允许其他医生使用同一医生号。如果计划将医生的年龄列的值范围设定为 1～150,则数据库不应接受 160。如果医生表中有部门号,该列存储医生所在的部门编号,则数据库应只允许接受医院中的有效部门编号。

数据完整性有 3 种类型:实体完整性(Entity Integrity)、引用完整性(Referential Integrity)、用户定义完整性(User-Defined Integrity)。

实体完整性指实体的所有主属性均不可取空值。由于一个实体映射到表中的一行，表的主键约束(PRIMARY KEY)保证了实体完整性。每个表只能有一个主键约束；用于主键约束的列的取值是不重复的(对由多列组成的主键，其各列取值组合不同)，而且不允许有空值(对由多列组成的主键，每个列均不允许有空值)。

对于东山县医院业务库中的医生表，“医生号”是主键，假设医生表中还没有医生号为501的医生(如果有，就先将其删除)，执行下列操作。

在医生表中插入如下语句：

```
INSERT INTO 医生 VALUES('501', '张力', '5')
```

再次插入：

```
INSERT INTO 医生 VALUES('501', '王丹', '5')
```

SQL Server 会出现错误信息，拒绝插入行。这是因为主键约束不允许键值重复。

再次插入：

```
INSERT INTO 医生 (姓名, 部门号)  VALUES('王丹','5')
```

SQL Server 会出现错误信息，拒绝插入行。这是因为主键约束不允许键值为空。

引用完整性也称为参照完整性。在输入或删除行时，引用完整性维护表之间已定义的关系。引用完整性通常通过外键(FOREIGN KEY)与主键之间参照实现。引用完整性保证键值在所有表中是一致的。这样的一致性要求不能引用不存在的键值。外键表的任意一个外键值必须等于主键表中所引用的主键的某个值或者为空。

例如，假设医生表中还没有定义对部门表中“部门号”的引用。假设部门表中没有6号部门。然后在医生表中插入：

```
INSERT  INTO 医生 VALUES('601', '潘辉','6')
```

由于医生表和部门表之间不存在外键约束，因此该行被成功插入医生表中。但事实上造成了数据不一致：某个医生在6号部门，而6号部门不存在。在医生表上建立外键约束可以避免这种“脏”数据。

假设医生表中定义了对部门表中“部门号”的引用，而且医生表中存在一行正在引用5号部门('501', '张力','5')。此时把部门表中5号部门“信息中心”的部门号改为8：

```
UPDATE 部门
SET 部门号 = '8'
WHERE 部门号 = '5'
```

SQL Server 会出现错误信息，拒绝更新行。如果在医生表中的外键列“部门号”上设置了级联更新，再次执行，则会看到不仅在部门表中5号部门的部门号改为8，而且医生表中所有部门号为5的行都改为8了。

用户定义完整性称为域完整性或语义完整性，包括检查约束、默认值定义、NOT NULL 和唯一约束等。

第3章

数据导入导出技术

假设用户正在使用 SQL Server 管理的数据库 A，那么把数据库 A 中的若干表复制到 SQL Server 数据库 B 中，或者把数据库 A 中的若干表复制到其他数据库管理系统中称为从 SQL Server 数据库中导出数据；反过来，把其他数据库管理系统中的表复制到数据库 A 中，或者把数据库 B 中的表复制到数据库 A 中，称为导入数据到 SQL Server 中。因为对数据库的所有操作都涉及权限控制，所以首先介绍一下数据库的安全管理。

3.1 用户以及授权

3.1.1 SQL Server 的安全体系结构

SQL Server 2008 为数据库和应用程序设置了 4 道安全防线：操作系统的安全防线、SQL Server 的运行安全防线、SQL Server 数据库的安全防线和 SQL Server 数据库对象的安全防线。用户要想获得 SQL Server 2008 数据库及其对象，必须通过这 4 道安全防线。4 道安全防线如图 3-1 所示。

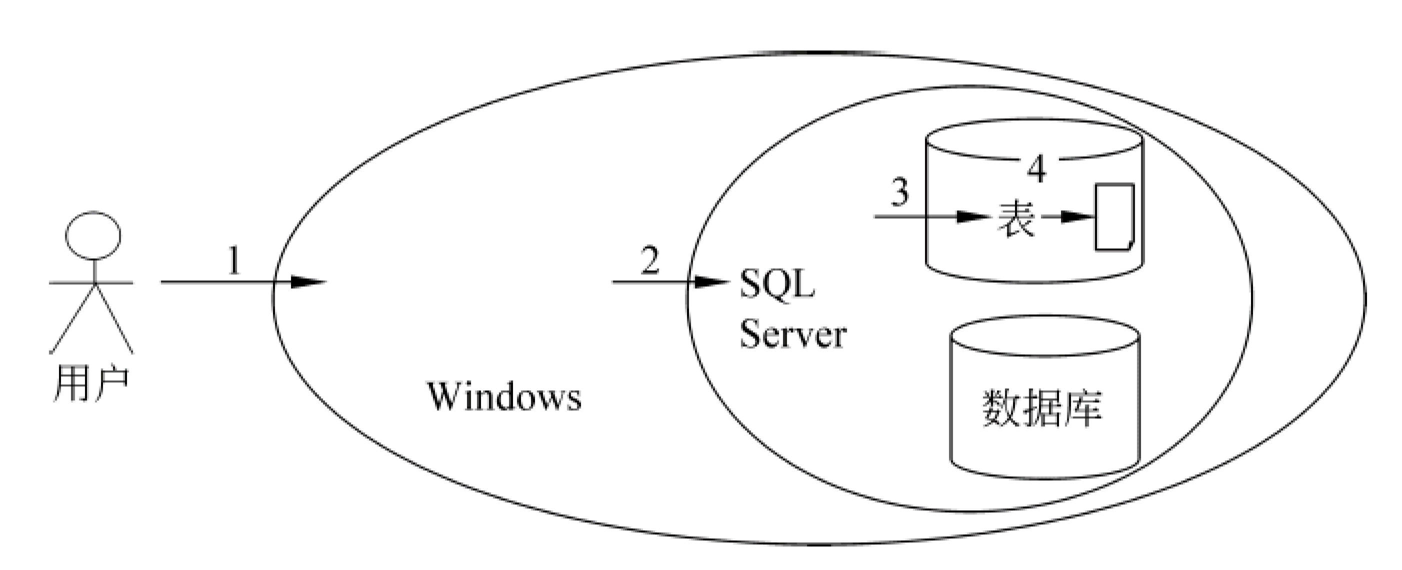


图 3-1 4 道安全防线

1）操作系统的安全防线

Windows（Windows 2013 Server 等）网络管理员负责建立用户组，设置账号并注册，同时决定不同的用户对不同系统资源的访问级别。用户只有拥有了一个有效的 Windows 登录账号，才能对网络系统资源进行访问。

2）SQL Server 的运行安全防线

SQL Server 通过登录账号设置创建附加安全层。用户只有登录成功，才能与 SQL Server 建立一次连接。

3）SQL Server 数据库的安全防线

SQL Server 的特定数据库都有自己的用户和角色，该数据库只能由它的用户或角色访问，其他用户无权访问其数据。数据库系统可以通过创建和管理特定数据库的用户和角色保证数据库不被非法用户访问。

4）SQL Server 数据库对象的安全防线

SQL Server 可以对权限进行管理。SQL Server 完全支持 SQL 标准的 DCL 功能，T-SQL 的 DCL 功能保证合法用户即使进入了数据库，也不能有超越权限的数据存取操作，即合法用户必须在自己的权限范围内进行数据操作。

3.1.2 安全认证模式

安全认证是指数据库系统对用户访问数据库系统时所输入的账号和口令进行确认的过程。安全认证的内容包括确认用户的账号是否有效、能否访问系统、能访问系统中的哪些数据等。

安全认证模式是指系统确认用户身份的方式。SQL Server 2008 有两种安全认证模式，即 Windows 安全认证模式和混合验证模式。

Windows 安全认证模式是指 SQL Server 服务器通过使用 Windows 网络用户的安全性控制用户对 SQL Server 服务器的登录访问。它允许一个网络用户登录到一个 SQL Server 服务器上时不必再提供一个单独的登录账号及口令，从而实现 SQL Server 服务器与 Windows 登录的安全集成。

混合验证模式下，用户可以使用 Windows 身份验证，也可以使用 SQL Server 标准账户登录。SQL Server 登录账号是独立于操作系统登录账号的，从而可以在一定程度上避免操作系统层上对数据库的非法访问。

无论哪种身份验证方式，采用 Windows 身份验证连接 SQL Server 总是可以的（只要该 Windows 用户是 SQL Server 的登录号）。

使用 SQL Server 2008 管理器设置安全认证模式的步骤如下：右击 SQL Server 服务器的名称，选择“属性”，再选择“安全性”选项卡；在“服务器身份验证”下根据需要选择“SQL Server 和 Windows 身份验证模式”或者“Windows 身份验证模式”。最后重新启动 SQL Server 服务。

3.1.3 用户管理

SQL Server 的用户分为服务器用户和数据库用户。

1. 服务器用户管理

1）登录账号

登录账号也称为登录用户或登录名，是服务器级用户访问数据库系统的标识。为了访问 SQL Server 系统，用户必须提供正确的登录账号，这些登录账号既可以是 Windows 登录账号，也可以是 SQL Server 登录账号。登录账号的信息是系统信息，存储在 master 数据库的 sysxlogins 系统表中，用户如需要有关登录账号的信息，可以到该表中查询。

SQL Server 2008 的登录账号 sa(System Administrator)在 SQL Server 系统中拥有全部权限,可以执行所有操作;登录账号"机器名\administrator"也拥有全部权限。

SQL Server 2008 有 3 种类型的登录账户:Windows 组、Windows 用户、标准。Windows 组和 Windows 用户是指 SQL Server 允许 Windows 的某个用户或某个组中的所有用户登录 SQL Server。标准账户是 SQL Server 自己的账户,使用自己的账户登录需要 SQL Server 身份验证方式运行在"SQL Server 和 Windows 身份验证模式"。

2) 查看登录账号

使用管理器可以创建、查看和管理登录账号。"登录账号"存放在 SQL 服务器的安全性文件夹中。当进入管理器,打开指定的 SQL 服务器,并选择"安全性"文件夹时,单击"登录名"可以看到当前数据库服务器的合法登录用户的一些信息。

3) 创建登录账号

创建一个登录账号的操作步骤为:右击"登录名"文件夹,在弹出的菜单中选择"新建登录名"选项后,会出现"新建登录"对话框界面,在其中填写相应信息即可。也可以通过此界面设定该登录用户的服务器角色和要访问的数据库,这样该登录账号同时也作为数据库用户。

例如,通过管理器建立标准登录账户"高林"的操作步骤如下:第一步,展开"安全性"结点,右击"登录名";第二步,从弹出的菜单中选择"新建登录名";第三步:在"登录名"文本框中输入"高林";第四步,单击"SQL Server 身份验证",输入密码;第五步,单击"确定"按钮。

通过管理器建立 Windows 组或 Windows 用户类型的登录账户 jack4 的操作步骤如下:右击登录名,从弹出的快捷菜单中单击"新建登录名",出现"登录名-新建"对话框。单击"登录名"文本框后的"搜索"按钮,弹出"选择用户或组"对话框,单击"高级"按钮,再单击"立即查找"按钮。在对话框的下面会出现用户名,选择用户名后单击"确定"按钮,关闭对话框,回到"登录名-新建"页面,名称项后面的文本框中出现所选择的 Windows 组或 Windows 用户,单击"确定"按钮即可。

4) 编辑或删除登录账号

单击"登录名"文件夹,在出现的显示登录账号的窗口中右击需要操作的登录号:选择"属性"便可对该账号已设定内容进行重新编辑(如修改密码);选择"删除"便可删除该登录账号。注意,删除前要断开该账号与服务器建立的活动连接。

可以把具有相同权限的某些用户设置成某一角色,这些用户称为该角色的成员。当对该角色进行权限设置时,其成员自动继承该角色的权限。这样,只要对角色进行权限管理,就可以实现对属于该角色的所有成员的权限管理,大大减少了工作量。

SQL Server 中有两种角色,即服务器角色和数据库角色。系统管理员(sysadmin)就是一种服务器角色。

使用管理器将登录账号添加到某一指定的服务器角色作为其成员的步骤如下:登录服务器后,展开"安全性"文件夹,单击"服务器角色"文件夹,右击某一角色,从弹出的菜单中选择"属性"命令,可以添加某些登录账号作为该角色的成员,也可以将某一登录账号从该角色的成员中删除。

2 数据库用户管理

1）数据库用户

数据库级用户是某个数据库的访问标识。数据库用户必须具有登录账号。登录用户只有成为数据库用户（或数据库角色）后才能访问数据库。用户与具体的数据库有关。例如，“东山县医院业务库”数据库中的用户账号“高林”不同于“东山县医院财务库”数据库中的用户账号“高林”。

每个数据库的用户信息都存放在系统表 sysusers 中，通过查看该表可以看到当前数据库所有用户的情况。在该表中，每一行数据表示一个 SQL Server 用户或 SQL Server 角色信息。创建数据库的用户称为数据库所有者（dbo），他具有这个数据库的所有权限。创建数据库对象的用户称为数据库对象的所有者，他具有该对象的所有权限。在每一个 SQL Server 2008 数据库中，至少有一个名称为 dbo 的用户。系统管理员 sa 是他所管理系统的任何数据库的 dbo 用户。

2）查看数据库用户

使用管理器可以创建、查看和管理数据库用户。每个数据库中都有“用户”文件夹。当进入管理器，打开指定的 SQL 服务器，并打开“数据库”文件夹，选定并打开要操作的数据库后，展开“安全性”，再单击“用户”文件夹就会出现用户信息窗口。通过该窗口可以看到当前数据库合法用户的一些信息。

3）创建新的数据库用户

展开数据库结点，右击“用户”文件夹，在弹出的菜单中选择“新建用户”命令后，会出现新建用户对话框界面，在“登录名”下拉列表框中选择预创建用户对应的登录名（即登录账号），然后在“用户名”的文本框中输入用户名。

4）编辑或删除数据库用户账号

单击“用户”文件夹，在出现的显示用户账号的窗口中右击需要操作的用户账号，选择“属性”命令，出现该用户的角色和权限窗口，可对该用户已设定内容进行重新编辑；选择“删除”便可删除该数据库用户。

进行上述操作需要对当前数据库拥有用户管理（db_accessadmin）及其以上的权限。

将数据库用户添加到某一数据库角色的步骤为：打开指定的数据库，单击“角色”文件夹，右击某个固定数据库角色，在出现的菜单中选择“属性”命令，单击“添加”按钮，则会出现该角色的非成员用户，按提示信息操作可以将他们添加到该角色中；选中某一用户后，单击“删除”按钮可以将此用户从该角色中删除。

在许多情况下，需要用户自定义数据库新角色。

使用管理器创建数据库角色的步骤为：在管理器中打开要操作的数据库文件夹，右击“角色”文件夹，并在弹出的菜单中选择“新建数据库角色”命令，则出现“新建数据库角色”对话框，按提示填写角色名称等相应信息后，单击“确定”按钮即可。

需要在“新建数据库角色”对话框的名称栏中输入新角色名；在成员栏中添加或删除角色中的用户。

3. 管理用户权限

可在管理器中授予数据库用户在数据库对象上的权限(如查询、修改、插入、删除等)。例如,授予数据库用户“高林”查询医生表的权限的步骤如下:第一步,单击“东山县医院业务库”的“用户”项;第二步,在右边用户列表中右击“高林”,在弹出的菜单中选择“属性”,单击“安全对象”页,单击“搜索”“确定”按钮,再单击“对象类型”,出现“选择对象类型”对话框,选择表;第三步,单击“浏览”按钮,出现“查找对象”对话框,选择表“医生”;第四步,返回“安全对象”页,勾选医生表上的“选择”权限;第五步,单击“确定”按钮。

在对话框的权限列表中,可以对每个对象进行授予、撤销或禁止权限操作。在权限表中,权限 SELECT(查询)、INSERT(插入)、UPDATE(更新)等安排在列中,每个对象的操作权用一行表示。在相应的复选框上,如果为“√”,则为授权;如果为“×”,则为禁止权限;如果为空白,则为撤销权限。单击复选框可改变其状态。

如果在管理器中登录 SQL Server 采用“Windows 身份验证”,这样默认的 Windows 用户是 Administrator,该用户在任何数据库中都具有 dbo(数据库所有者)角色,所以具有管理数据库其他用户的权限。

权限两个字,一个权力,一个限制。就是哪些人可以对哪些资源做哪些操作。在 SQL Server 中,“哪些人”“哪些资源”“哪些操作”分别对应 SQL Server 中的 3 个术语:主体(Principals,如高林)、安全对象(Securables,如表)和权限(Permissions,如 SELECT),而权力和限制则对应了 SQL Server 中的 GRENT 和 DENY。

“主体”是可以请求 SQL Server 资源的实体。主体可以是个体、组或者进程。主体可以按照作用范围被分为三类:Windows 级别主体、服务器级别主体和数据库级别主体。

Windows 级别的主体包括 Windows 域登录名和 Windows 本地登录名。

SQL Server 级的主体包括 SQL Server 登录名和服务器角色。

数据库级的主体包括数据库用户和数据库角色以及应用程序角色。

登录名是服务器级别的主体,但无论是上述哪个层级的主体,因为需要登录到 SQL Server 实例,所以每一个层级的主体都需要一个与之对应的登录名。对于 Windows 级别的主体来说,Windows 用户会映射到登录名。对于数据库级别的主体来说,其用户必须映射到登录名中,而登录名可以不映射到数据库用户。

数据库用户是数据库级别的主体,被用于访问数据库层面的对象。默认情况下,每个数据库都有 4 个内置用户:dbo、guest、INFORMATION_SCHEMA 和 sys。

dbo 用户是 Database Owner 的简称,如果说 SA 是实例级别的统治者,那 dbo 就是数据库级别的统治者。这个用户不能被删除,每一个属于 sysadmin 的服务器角色都会映射到数据库的 dbo 用户。

guest 用户是一个来宾账户,这个账户允许登录名没有映射到数据库用户的情况下访问数据库。默认情况下,guest 用户是不启用的,可以通过如下代码启用或不启用 guest 用户。

```
--允许 guest 用户连接权限
  GRANT CONNECT TO guest
```

```
--收回 guest 用户的连接权限
  REVOKE CONNECT FROM guest
```

角色便于对主体进行管理。属于某个角色的用户或登录名拥有相同的权限。当具有相同职责的数据库用户很多时，一般首先为这些用户定义一个共同的“角色”，然后为这个角色授予权限，最后再去创建数据库用户并将这个角色与之关联起来。

public 角色是一种内置角色，其权限可以被调整。可以将 public 角色理解为访问数据库或实例的最小权限。public 拥有的权限自动被任何主体继承。用户自定义角色是按照用户自己的需求组成的角色，由用户创建。

架构（Schema）是一个命名空间，该空间拥有一些数据库层级的对象。默认架构是 dbo 时，如果被访问的对象的架构也是 dbo，可以在被访问的对象前省略架构名称：

```
SELECT * FROM Customer
```

否则，必须有架构名称：

```
SELECT * FROM dbo.Customer
```

3.1.4 数据控制语句

用户权限主要包括数据对象和操作类型两个要素。定义用户的存取权限称为授权（GRANT），通过授权规定用户可以对哪些数据进行什么样的操作。表 3-1 列出了不同类型的数据对象的操作权限。

表 3-1 不同类型的数据对象的操作权限

类别		操作权限
数据对象	表、视图	SELECT、INSERT、UPDATE、DELETE
	表、视图中的列	SELECT、UPDATE
语句		CREATE DATABASE、CREATE TABLE、CREATE VIEW

授权 GRANT 的语法：

```
GRANT <权限 1> [,<权限 2>,…] [ON <对象名称>] TO <用户 1> [,<用户 2>,…]
```

其功能是将指定数据对象的指定权限授予指定的用户。下面通过例子理解 GRANT 语句的数据控制功能。假设已经存在用户“高林”，那么把对部门表中的“名称”列的修改权限以及部门表的查询权限授予用户“高林”的语句如下。

```
GRANT UPDATE(名称), SELECT ON 部门 TO 高林
```

把在数据库中建立表的权限授予用户“高林”的语句如下。

```
GRANT CREATE TABLE TO 高林
```

回收（REVOKE）权限的语法：

```
REVOKE <权限 1> [,<权限 2>…]  [ON<对象名>]
FROM <用户 1> [,<用户 2>…]
```

其功能是把已经授予指定用户的指定权限收回。例如,把用户“高林”修改部门名称的权限收回的语句如下。

```
REVOKE UPDATE(名称) ON 部门 FROM 高林
```

3.2 从 SQL Server 数据库导入表

3.2.1 利用数据库的分离/附加功能实现数据导入

“分离”和“附加”是互逆的两个操作。“分离数据库”是断开数据库与 SQL Server 的被管理与管理关系,但并不在磁盘上删除组成数据库的数据文件和日志文件。使用分离/附加功能实现数据采集主要有 3 个步骤:从被审单位数据库服务器上分离所需的数据库;审计人员将该数据库的数据文件和日志文件复制到自己的移动存储介质上;将数据库附加到自己本地计算机的 SQL Server 数据库服务器上。

在分离数据库之前,应该先查看要分离的数据库包含的全部数据文件和日志文件,以及这些文件的存放位置和文件名,以便于分离后对这些文件进行复制。查看数据库包含的全部数据文件和日志文件可以在管理器中实现,也可以利用系统存储过程实现。

在管理器中右击待分离的数据库,从弹出的菜单中选择【任务】|【分离】,如果还有其他连接(例如,在 SQL 编辑器中正在使用该数据库),则在“分离数据库”对话框中设置“删除连接”,之后单击“确定”按钮即可实现分离。

分离结束后,将涉及的数据文件和日志文件复制到自己的移动硬盘或自己的计算机中,然后利用附加功能把数据库附加上。数据库附加步骤如下。

在对象资源管理器中右击“数据库”,从弹出的菜单中选择“附加”。在“附加数据库”对话框中需要指定数据库主文件的文件名,单击“添加”按钮,在出现的对话框中选择数据库主文件的文件名,单击“确定”按钮。

回到附加数据库页面,数据库文件和日志文件自动列出来,在“附加为”下面设置数据库逻辑名,可以和原来的名称不一样。由于数据文件和日志文件在源服务器上的路径一般不同于所要附加的服务器的路径,所以通常需要修改“mdf 文件位置”,最后单击“确定”按钮即可。

3.2.2 直接复制数据库中的文件

先将被审单位的 SQL Server 服务停止,然后将所需的数据文件和日志文件复制到自己的移动硬盘或自己的计算机中,再将被审单位的 SQL Server 服务启动起来。在自己的计算机上,利用附加功能可以将数据库添加进来。

3.2.3 备份/还原

使用备份/还原的办法采集数据主要有 3 个步骤:在被审单位数据库服务器上完整

备份所需的数据库；把备份文件复制到移动介质上；把备份文件复制到审计人员本地计算机的硬盘上，并恢复到本地数据库服务器中。

备份的步骤如下：在 SQL Server 管理器里选中要转移的数据库，按鼠标右键，选择【任务】|【备份】。在“备份数据库”对话框中，保持其他选项不变，在“目标”项中单击“添加”；在弹出的菜单中选择按文件保存备份，在“文件名”后输入要保存到的路径和文件名。文件名的扩展名约定为.bak。单击“确定”按钮，关闭“选择备份目标”对话框，回到“备份”对话框，再次单击“确定”按钮，开始备份进程。

备份文件可以随便迁移到任何地方，包括另外的 SQL Server 服务器。

在本地计算机上启动管理器，下面的步骤把备份文件恢复到一个新数据库“test”中：右击数据库文件夹，选择“还原数据库”；在“还原数据库”对话框中输入“目标数据库”的数据库名字“test”，单击“源设备”；单击“源设备”右侧的按钮，单击“添加”按钮；找到要导入的备份数据库文件名，单击“确定”按钮；单击“确定”按钮关闭“指定备份”对话框。

单击“还原数据库”中的“选项”选择页，保持数据文件和日志文件的逻辑名不变。由于备份文件中记载的是被审单位的数据文件路径，所以“还原为”栏中显示了被审单位的数据文件全路径名。该路径一般与审计人员本地计算机路径不同，所以通常在“还原为”列中修改数据文件和日志文件的物理文件名为本地路径名，单击“确定”按钮，完整的数据库导入就成功了。

如果还原到本地服务器中已经存在的数据库上，则需要勾选“覆盖现有数据库”。

3.2.4 导入导出

假设能够将审计人员的笔记本连入被审单位的计算机网络，并能够访问所需的数据库。下面的步骤演示了如何从被审单位数据库中把所需的表导入本地数据库中。

首先，在本地计算机的 SQL Server 中创建一个空的数据库，如名字为“我的数据库”，准备接收表对象。

然后，将本地计算机连入网络，设置适当权限，使其能够访问数据源所在的 SQL Server 服务器。假如该服务器的名字为 lx080。

在管理器中右击“我的数据库”，选择【任务】|【导入数据…】，出现“SQL Server 导入和导出向导”。在“选择数据源”对话框中设置服务器为 lx080，数据库为东山县医院财务库；在“选择目标”对话框中设置服务器为 local，数据库为“我的数据库”；在“指定表复制或查询”对话框中选择“复制一个或多个表或视图的数据”。在“选择源表或源视图”对话框中单击“源”前面的方框，选择所有表和视图。单击“下一步”按钮，选择“立即运行”。

3.3 从其他数据库导入表

在审计实践中，经常需要将各种来源的数据（如 Access 数据库或 ASCII 文本文件）移植到 SQL Server 2008 数据库中。这种把不同来源的数据迁移到 SQL Server 2008 数据库中的过程就是数据导入。

3.3.1 把 Access 数据导入 SQL Server

假设已经从被审单位获取一个 Access 数据库：C：\data\东山县医院财务数据.mdb。把它通过 SQL Server 导入导出向导导入 SQL Server 中有两个途径：Access 驱动途径和 ODBC 驱动途径。

如果存在可供 SQL Server 导入导出向导直接使用的驱动，则可以不使用 ODBC 这一层驱动软件，而直接使用源数据库的驱动软件。

首先看 Access 驱动途径的导入。

假设在 SQL Server 中已经存在一个准备接收 Access 数据库中表的空数据库“我的数据库”（如果没有，则新建一个空数据库）。直接使用 Access 驱动途径的步骤如下：①在 SQL Server 管理器中的“我的数据库”上右击，选择“任务”，继续选择“导入数据”，出现 SQL Server 数据导入导出向导。②在“选择数据源”对话框中选择 Microsoft Access，然后键入东山县医院财务数据.mdb 数据库(.mdb 为文件扩展名)的文件名或通过浏览查找该文件。③在“选择目标”对话框中选择 Microsoft OLE DB Provider for SQL Server，选择数据库服务器，然后选择所用的身份验证方式。在“指定表复制或查询”对话框中选择“从数据源复制表和视图”。④在“选择源表和源视图”对话框中，左侧列出了 Access 数据库中的所有表，单击表前的检查框表示将其导入。该名字就会出现在右侧列表。默认导入的表名保持不变。若需要改名，则在名字上单击，出现插入点后修改成新的名字即可。例如，把“会计科目表”改名为“会计科目”。单击“下一步”按钮，选择“立即运行”。导入视图时会把源视图里所有的真实数据导入成一个新表，而不是视图。

从 Access 数据库导入 SQL Server 的导入过程如图 3-2 所示。

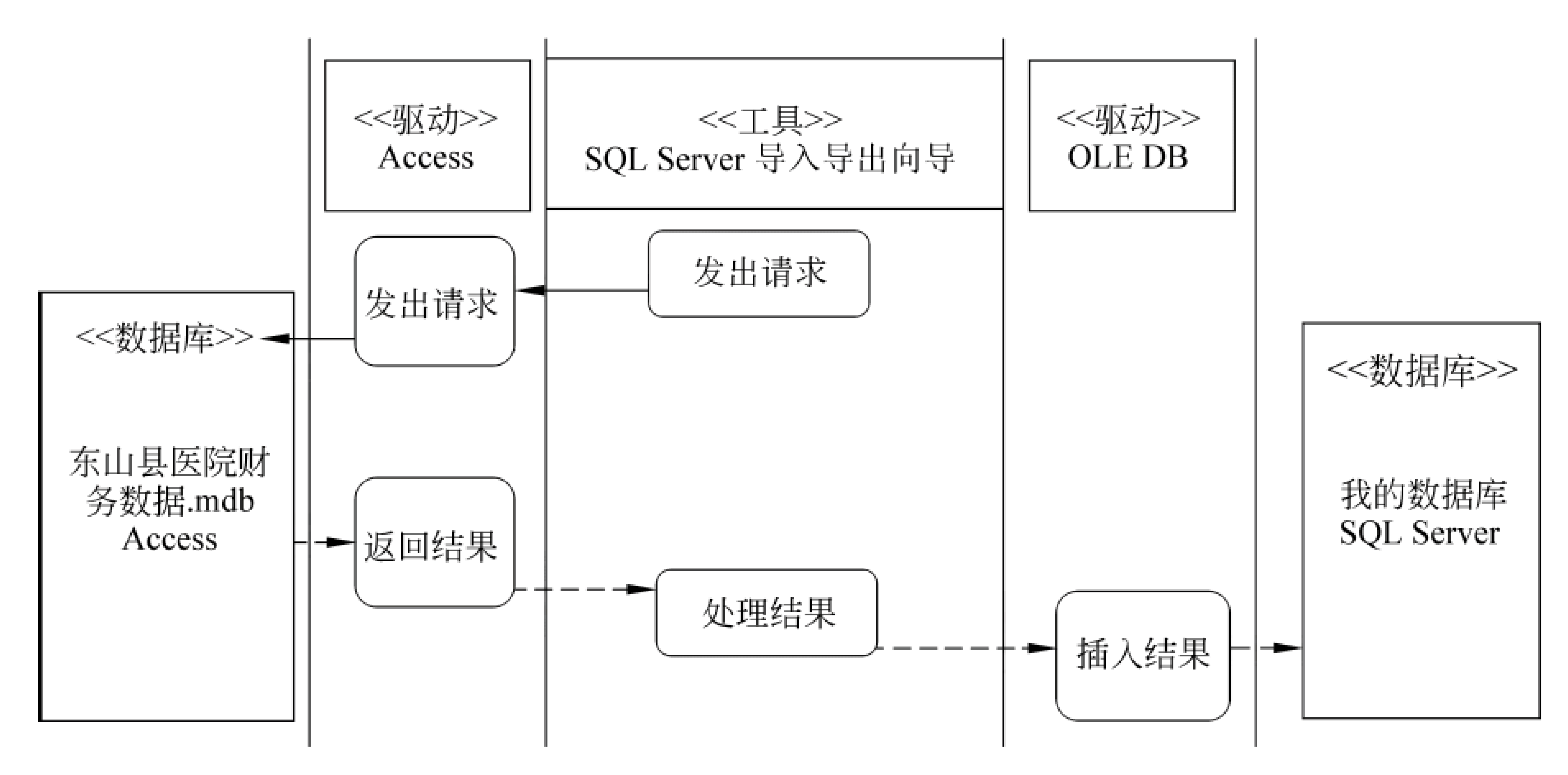


图 3-2 从 Access 导入 SQL Server 的导入过程

由于 SQL Server 中的数据类型与 Access 中的数据类型不同，所以导入导出向导按照默认的数据类型对应进行转换，见表 3-2。

表 3-2　SQL Server 导入导出向导默认的数据类型对应

Access	SQL Server	是否允许空
文本(255)	nvarchar (255)	NULL
备注	nvarchar(max)	NULL
数字(字节)	tinyint	NULL
数字(整型)	smallint	NULL
数字(长整型)	int	NULL
数字(单精度)	real	NULL
数字(双精度)	float	NULL
日期/时间	datetime	NULL
货币	money	NULL
是/否	bit	NOT NULL
自动编号	int	NOT NULL

如果要更改默认的数据类型，一种方法是转换后在 SQL Server 数据库中通过“设计表”对话框进行修改，另一种方法是转换时就进行修改。例如，把 nvarchar 改为 varchar 类型，如图 3-3 所示。



图 3-3　更改默认的数据类型映射

选择源表和视图时，如果想修改会计科目表的数据类型，就单击“编辑映射”。在“列映射”对话框中选择“目标”中“类型”的下拉列表，从中选择需要的数据类型，如 varchar，把大小由默认 254 改为 9。

接下来讨论 ODBC 途径的导入。

在 SQL Server 2008 中，导入导出向导支持的外部数据源较多，如图 3-4 所示。其中有些是新增的。

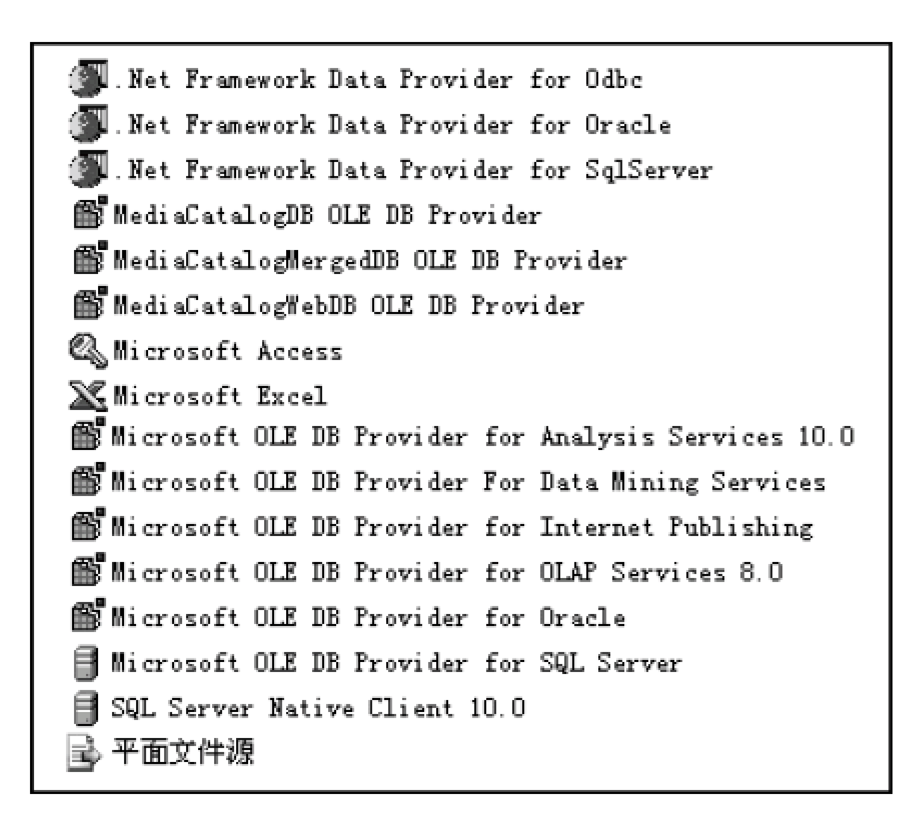


图 3-4　数据源

这些数据源可以分为三类。一是.NET Framework Data Provider 提供的访问方式，如.NET 为 ODBC、Oracle、SQL Server 提供了访问，安装.NET 后，即可使用这些访问方式。二是 OLE DB Provider 提供的访问方式。安装 Visual Studio 等开发工具后，即可使用这些访问方式。三是特定应用系统的驱动，如 Excel，安装 Excel 之后自带。

ODBC 途径分为 3 个步骤：第一步，建立访问 Access 数据库(C:\data\东山县医院业务数据.mdb)的 ODBC 数据源；第二步，建立接收 Access 数据库(C:\data\东山县医院业务数据.mdb)中的表或者视图的 SQL Server 数据库；第三步，运行 SQL Server 导入导出向导(导入和导出)，执行导入。ODBC 途径的数据流如图 3-5 所示。

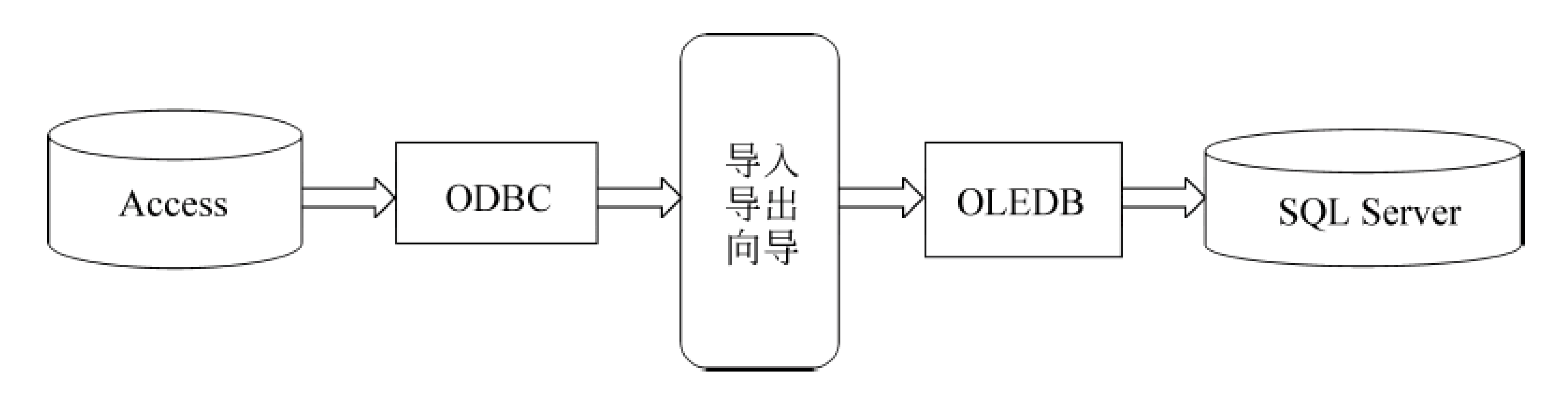


图 3-5　ODBC 途径的数据流

具体步骤如下。

第一步，建立 ODBC 数据源，让 SQL Server 导入导出向导通过该数据源访问东山县医院财务数据.mdb。

(1) 双击“控制面板”的“管理工具”中的“数据源(ODBC)”，会看到“ODBC 数据源管理器”对话框，如图 3-6 所示。注意，该对话框中当前显示的选项卡是“用户 DSN(数据源

名)”选项卡。用户数据源仅对当前用户可见，而且仅能用于当前计算机上；对系统级数据库来说，使用“系统 DSN”选项卡。系统数据源对当前计算机上的所有用户可见；对文件级数据源(从严格意义上说，它不是数据库)，使用“文件 DSN”选项卡。文件数据源可以由安装了相同驱动程序的用户共享。

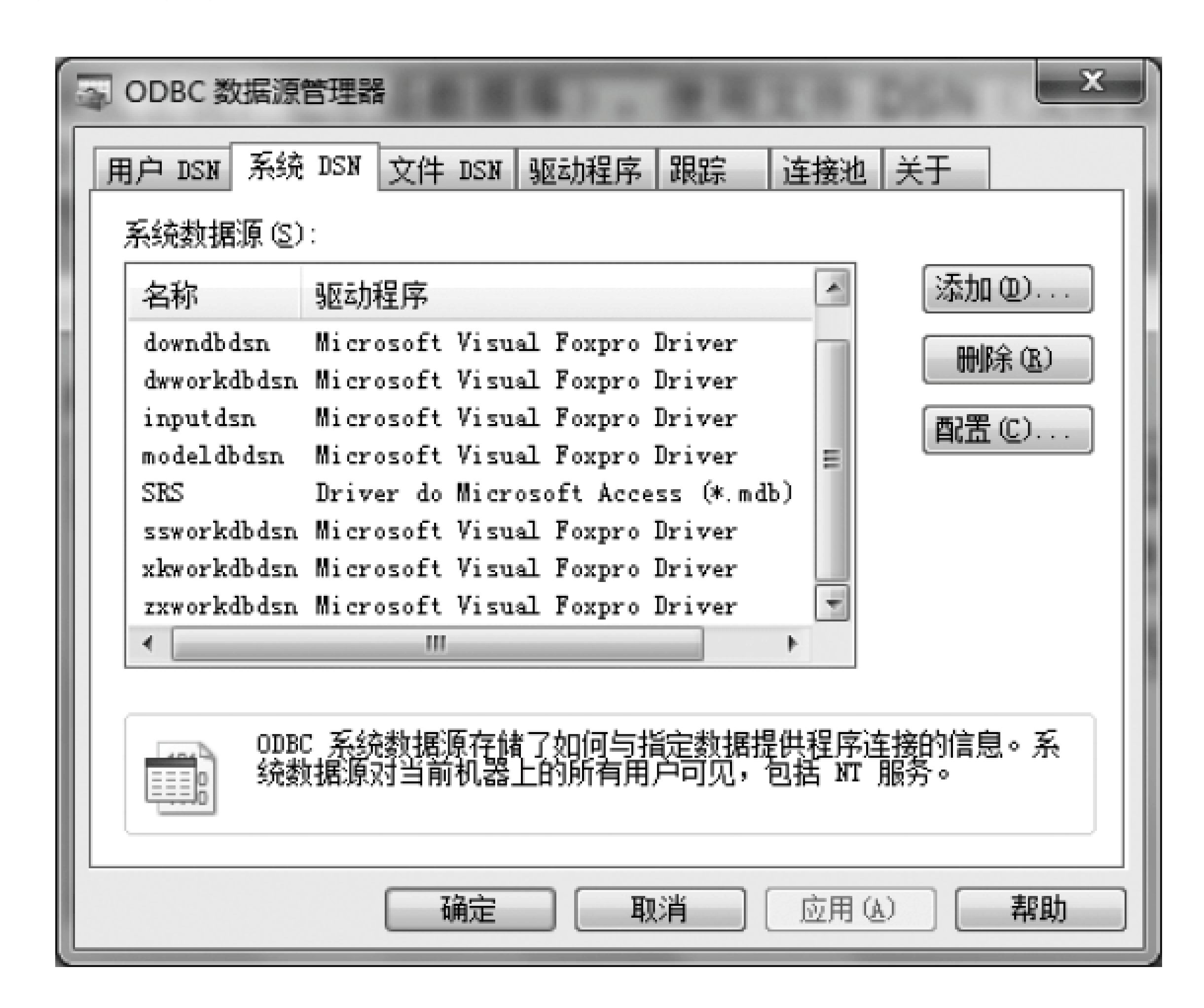


图 3-6　查看系统数据源

对本地数据库来说，通常要在“用户 DSN”选项卡上创建一个项；对远程数据库来说，则要在“系统 DSN”选项卡上创建一个项。任何情况下都不能在“用户 DSN”和“系统 DSN”选项卡上创建同名的项。

(2) 选择“系统 DSN”选项卡，单击“添加”按钮会看到“创建新数据源”对话框，如图 3-7 所示。



图 3-7　创建新数据源

(3) 选择一个数据源的驱动程序。对本例选择 Microsoft Access Driver(*.mdb, *.accdb)，单击“完成”按钮，出现“ODBC Microsoft Access 安装”对话框，如图 3-8 所示。



图 3-8 “ODBC Microsoft Access 安装”对话框

(4) 在“数据源名”文本框中输入数据源名称。一定要选择意义明确但又不过于冗长的名称。输入 Dongshan,因为最终要创建一个与东山县医院有关的数据库的链接。

(5) 在“说明”文本框中输入一段说明性文字。可以让这个项比上一个项稍长一些,因为它描述数据库的用途,这里输入 Accounting。

(6) 单击“选择”按钮,会看到一个“选择数据库”对话框,可以在那里选择一个现有的数据库。ODBC 驱动程序会自动选择正确的文件扩展名。

(7) 单击“确定”按钮关闭“ODBC Microsoft Access 安装”对话框。应该看到,新的设置项已经添加到“ODBC 数据源管理器”对话框中,如图 3-9 所示。如果今后要为数据库更改这些设置,选择它并单击“配置”按钮即可。如果要删除数据源,选择 DSN 并单击“删除”按钮即可。



图 3-9 添加 Dongshan 后的系统数据源

第二步,参见在 SQL Server 中通过管理器创建数据库的步骤,创建一个空的数据库准备存放导入的表。不妨把数据库命名为 TEST。

第三步,通过选择“导入导出向导”实现导入。

（1）在“对象资源管理器”中右击 TEST 数据库，然后选择“任务”，再选择“导入数据”。

（2）在“选择数据源”对话框中选择“.Net Framework Data Provider for Odbc”数据源；在 Dsn 右侧的文本输入框中输入已经在 Windows 中设置好的数据源名 Dongshan，单击“下一步”按钮，如图 3-10 所示。

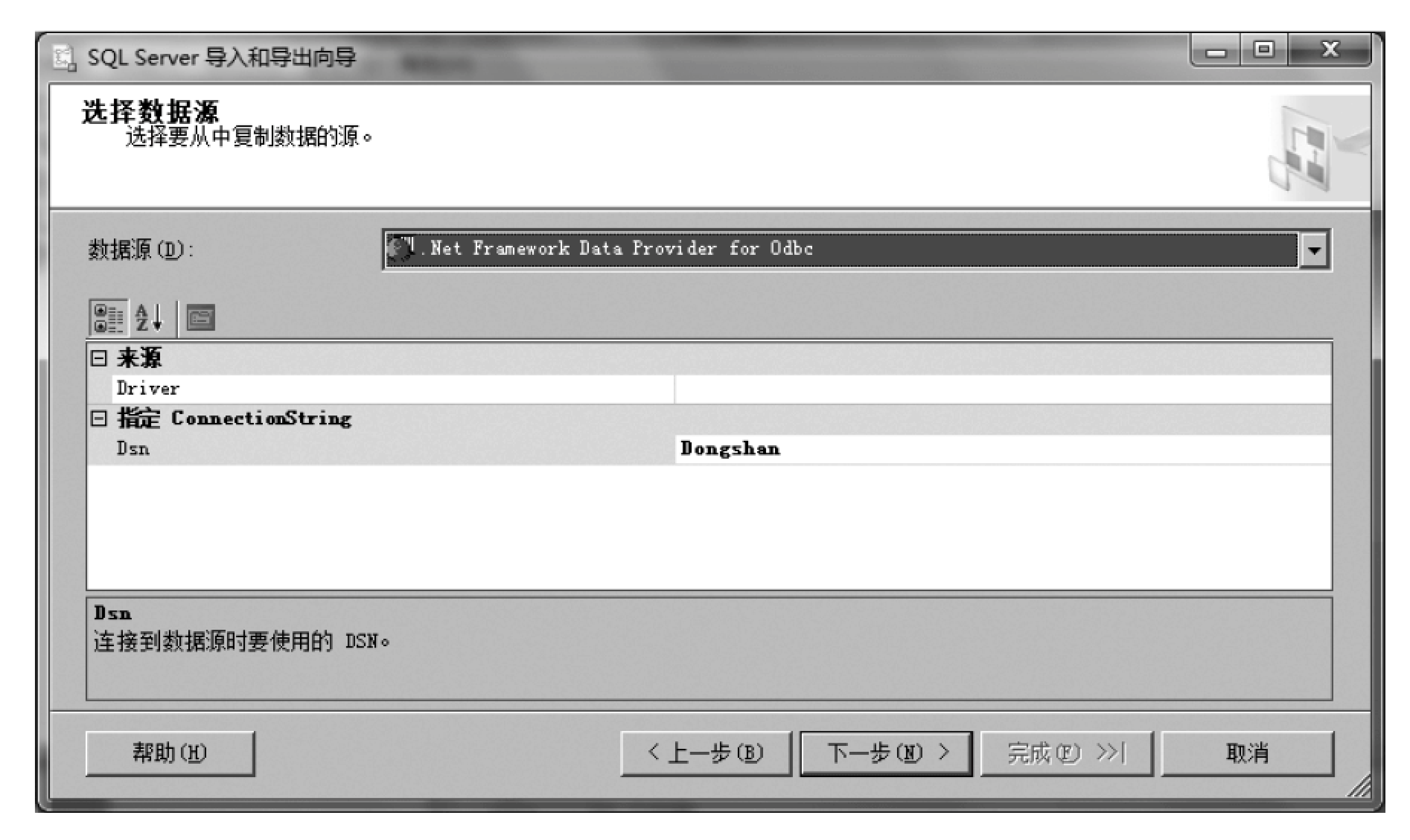


图 3-10　选择数据源

（3）选择目标如图 3-11 所示。由于从 TEST 数据库出发启动了导入导出向导，所以默认的目标是 TEST 数据库，保持其不变，单击“下一步”按钮。



图 3-11　选择目标

（4）选中“复制一个或多个表或视图的数据”，如图 3-12 所示，单击“下一步”按钮。

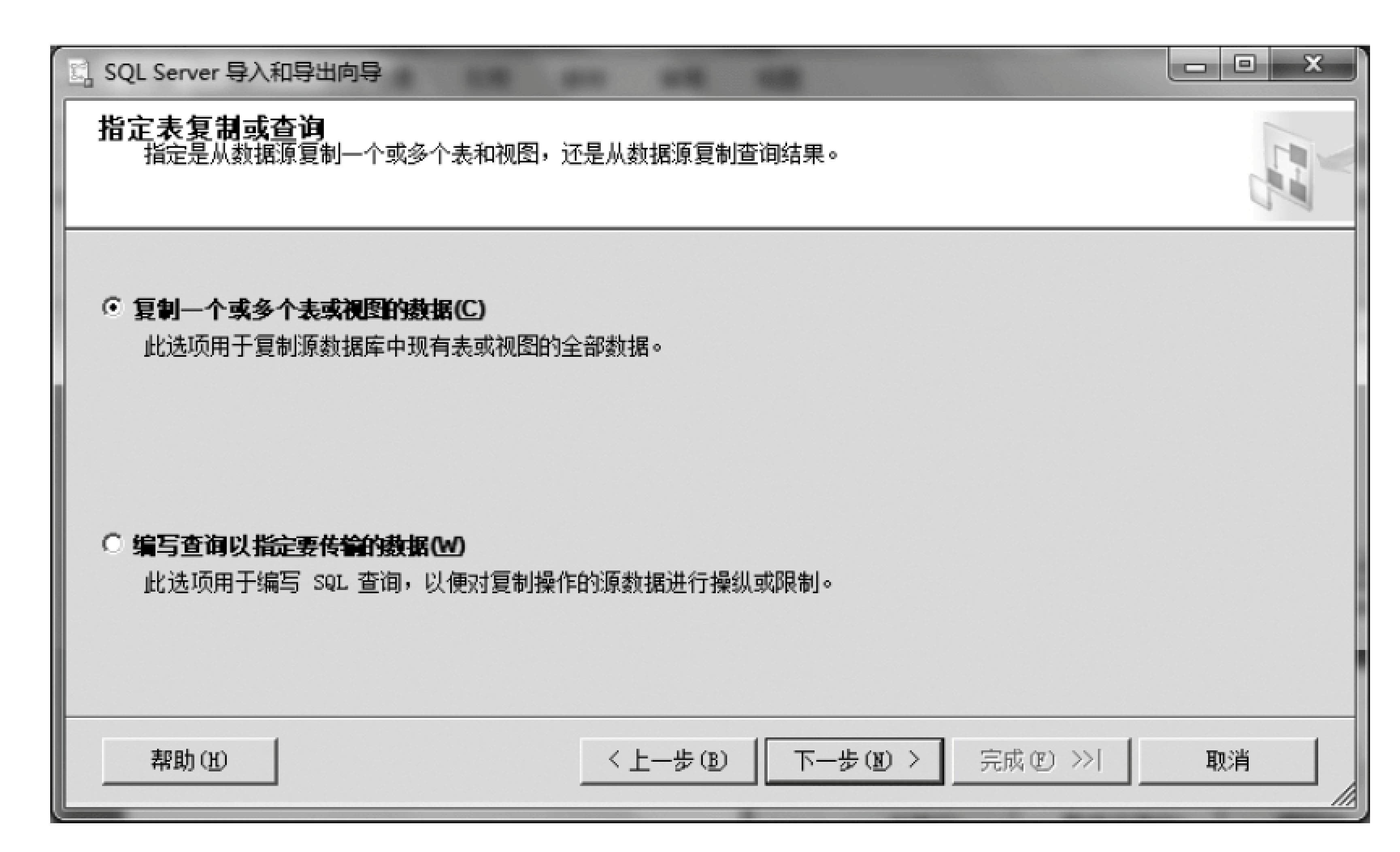


图 3-12 指定表复制或查询

（5）在“选择源表和源视图”窗口左侧勾选表和视图后，单击“下一步”按钮，如图 3-13 所示。



图 3-13 选择源表和源视图

（6）勾选“立即运行”，单击“下一步”按钮。确认要做的事情，单击“完成”按钮。

ODBC 途径的导入各部件间的调用关系如图 3-14 所示。SQL Server 导入导出向导的数据访问请求通过 ODBC 驱动软件交给了 Access 驱动软件，由 Access 驱动实施具体读取，并将结果通过 ODBC 交给 SQL Server 导入导出向导。SQL Server 导入导出向导再通过使用 SQL Server 的 OLE DB 驱动软件把数据写入 SQL Server 的数据库。注意，为了方便起见，ODBC 途径的数据导入仍然用了 Access 数据库。显然，对于 Access 来

说，这种途径比直接导入 Access 数据库复杂。但是，对于非 Office 数据库，如 DBASE 数据库，则只能使用此途径。

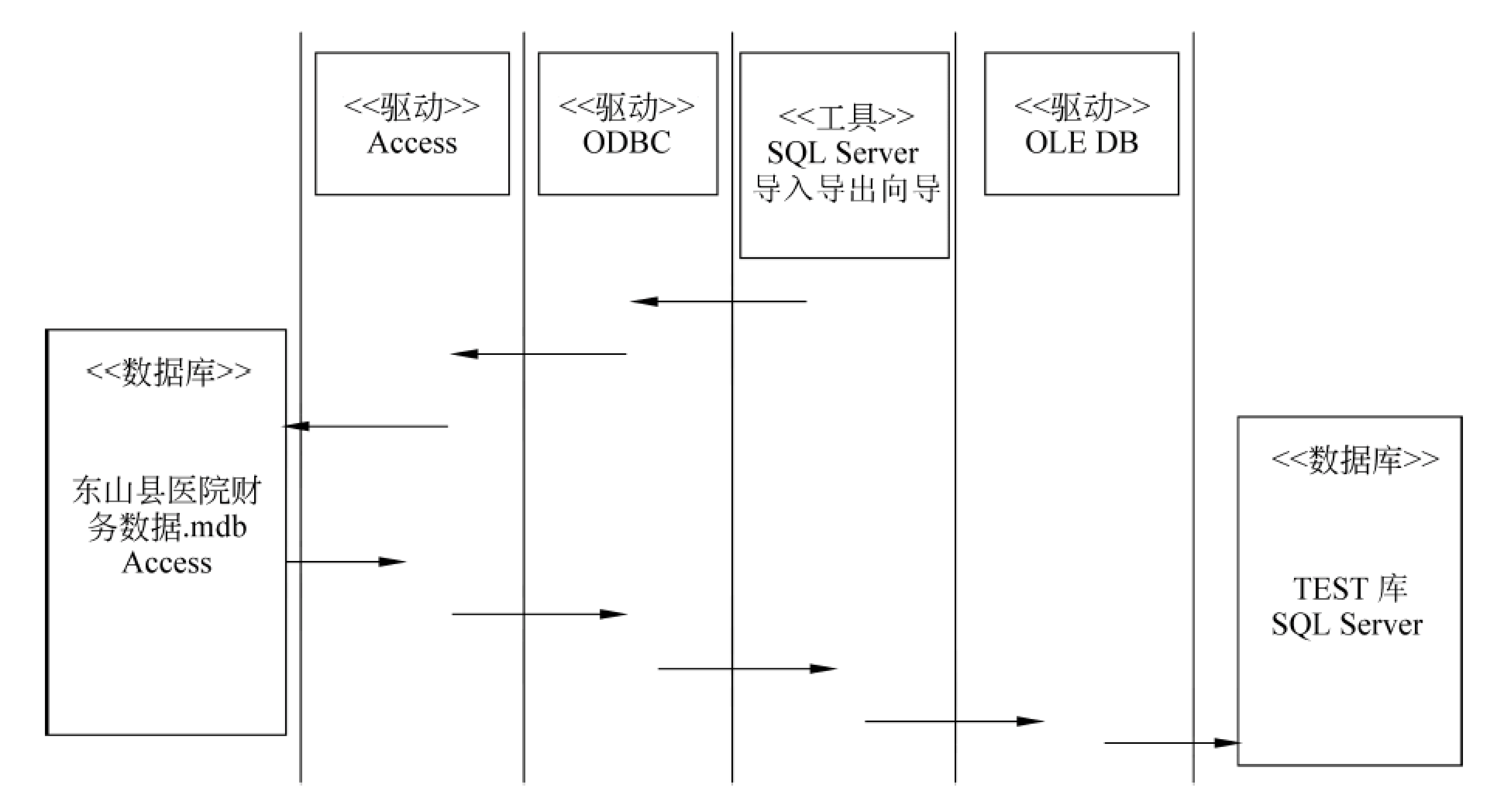


图 3-14　ODBC 途径的导入各部件间的调用关系

ODBC 数据源的概念如图 3-15 所示。

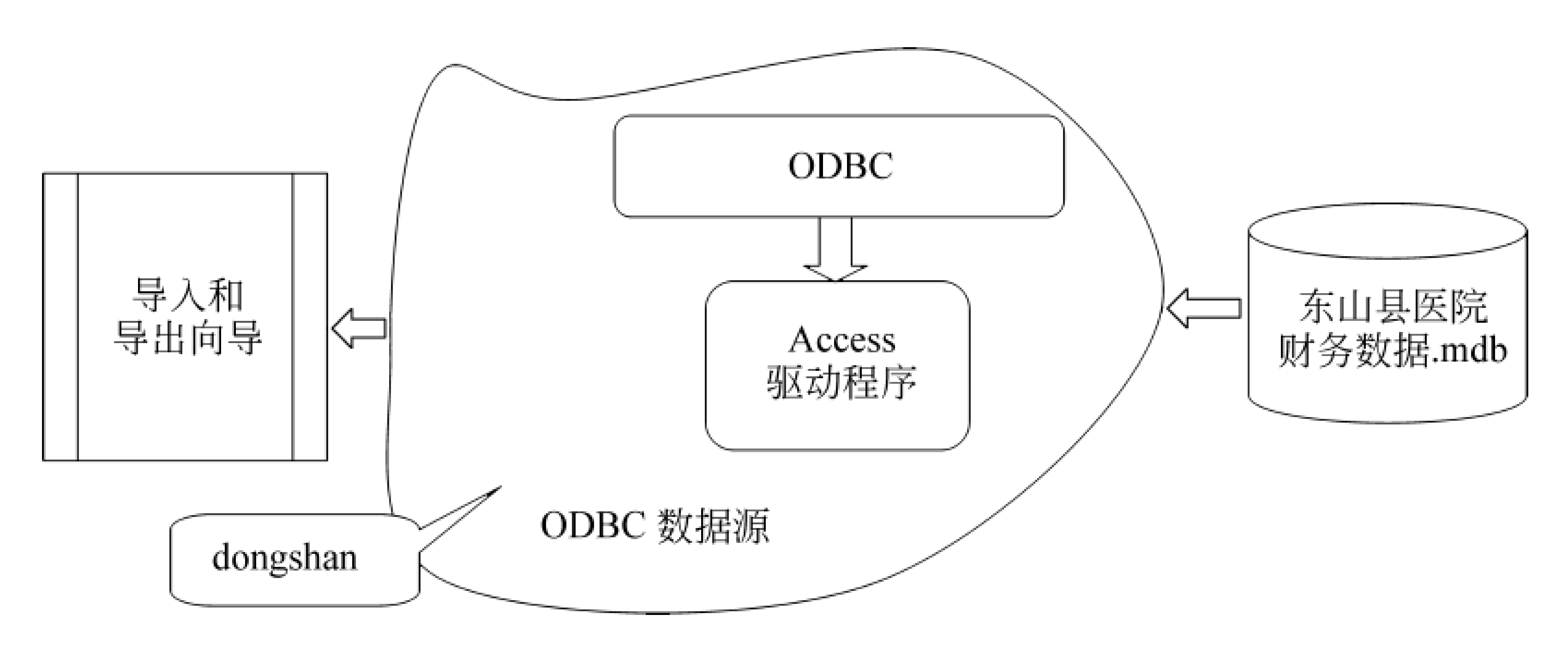


图 3-15　ODBC 数据源的概念

大多数数据库厂商都提供了用于 ODBC 的驱动程序，但是不一定提供用于某个第三方软件（如 SQL Server 导入导出向导）的驱动程序。所以，ODBC 的途径对于访问大多数数据库系统来说是一种通用的途径。

3.3.2　把文本文件导入 SQL Server

一般来说，用于数据交换的文本文件有两种格式：字段由分隔符分隔和字段固定宽度。分隔符可以是逗号、制表符等。例如，下面是一个以逗号分隔字段的文本文件银行.txt 局部。

```
银行编码,业务代码,名称,利率,开始日期
500,100,90 年五年期财政债券利率,8.31,01/10/1997
500,200,开发银行债券利率(12.5%),10.41,01/10/1997
500,201,开发银行债券利率(14%),11.64,01/10/1997
```

500,1100,单位活期存款,1.2,07/01/1998
...

下面是一个以制表符分隔字段的文本文件(贷款利率表 TAB)局部。

银行编码	业务代码	名称	利率	开始日期
500	100	90年五年期财政债券利率	8.31	01/10/1997
500	200	开发银行债券利率(12.5%)	10.41	01/10/1997
500	201	开发银行债券利率(14%)	11.64	01/10/1997
500	1100	单位活期存款	1.2	07/01/1998

...

由于制表符是控制字符,控制了文本的位置,所以有时从视觉上无法区分是单个制表符,还是多个空格。可以在 Microsoft Word 中打开该文件,然后打开"工具"菜单的"选项"菜单项,在"视图"选项卡的"格式标记"区勾选"制表符",就可以观察制表符号了。

字段固定宽度的文本文件中每一行的数据都包含相同的字符个数,即每一行的宽度相等。例如,下面的文本文件序号字段固定 3 个字符宽度,商品名称固定 10 个字符宽度,数量固定 3 个字符宽度,单价和金额固定 9 个字符宽度。

序号	商品名称	数量	单价	金额
01	隆盛开心果	2	37.00	74.00
02	雀巢咖啡伴侣(瓶)	1	23.20	23.20
03	双汇玉米热狗	1	8.70	8.70
04	衡水老白干淡雅	1	59.00	59.00

...

下面演示把银行.txt 导入到 TEST 数据库的过程。

(1) 运行导入和导出数据工具,在弹出的"导入和导出向导"窗口单击"下一步"按钮,在"选择数据源"对话框的数据源下拉列表中选择"平面文件源",并在文件名文本框中输入文件名或者打开"文件"对话框,选择所要的文件。

在"格式"设置中,保持默认的"带分隔符"格式不变。文本文件中的行或列的长度可以是固定的,也可以用特殊的字符分隔。

当将数据从文本文件复制到新的 SQL Server 表时,列的大小默认为 varchar(50)。如果需要调整列的大小设置,请单击"高级"选项卡修改列。

"带分隔符"指数据在文件内按字段对齐,每个字段均由相同的分隔符分隔。"固定字段"指数据在文件内以等宽按字段对齐。文件内的字段对所有数据行宽度都相等。然而,在同一行中,各字段宽度都可以不同。

"文本限定符"指数据文件中用来界定文本的字符标记。单击下列文本限定符之一:双引号、单引号、无,也可以输入字符,用作文本限定符。如果文本没有被分隔,而文件不是"固定字段"文件,则该属性可能设置为默认值"双引号"。

"标题行分隔符"指文件中各行以某种字符序列彼此分离。单击下列分隔符之一:CR(回车)、LF(换行符)、;(分号)、,(逗号)、制表符、|(垂直条),也可以输入字符,用作行分隔符。如果文件是固定字段文件,就选择"无"选项。

“要跳过的标题行数”指从文件开头起不想复制的行数。该字段与“第一行含有列名称”字段一起使用。如果没有勾选“在第一个数据行中显示列名称”复选框，则跳过的行数等于这里输入的数字。如果勾选了“在第一个数据行中显示列名称”复选框，则跳过的行数不包括标题行。

“在第一个数据行中显示列名称”指文本文件的第一行含有列标题，而非数据。

向导根据此设置，能够自动识别列。单击左侧的“列”标签，就可以观察显示的结果。本例中，向导自动识别文本文件中共有 5 行，其中第 1 行为标题行，其余 4 行为数据行。共有 5 列，各个列的名字显示了出来。

根据文本文件中具体的列间分隔符进行选择。如果没有，则要修改文本文件，在各列间插入制表符。

（2）单击“下一步”按钮，选择目标服务器和数据库。

（3）在“选择源表和源视图”对话框中，保持默认设置。

当将数据从文本文件复制到新的 SQL Server 表时，列的数据类型默认为 Nvarchar(50)。如果需要调整列的数据类型设置，请单击“编辑映射”按钮修改目标列，最后单击“完成”按钮。

对于固定列宽的文本文件，还可以先将其导入 Access 数据库中，然后再从 Access 数据库中导入到 SQL Server 数据库中。下面演示了从固定列宽的文本文件导入到 Access 数据库的步骤。

第一步，在 Access 中新建一个空的数据库，不妨命名为“超市.mdb”。注意记住该数据库在磁盘上的位置。

第二步，在 Access 菜单栏中选择“外部数据”菜单，单击“文本文件”工具按钮，出现“导入”对话框，在“导入”对话框中首先通过“文件类型”下拉列表框指定文本文件类型，然后选择准备导入的文本文件，如固定宽度.txt，最后单击“确定”按钮。

在“导入文本向导”对话框中单击“高级…”按钮，出现如图 3-16 所示的“设置导入规格”对话框。



图 3-16　设置导入规格

图 3-17 是向导猜测的结果。单击“确定”按钮关闭对话框。

图 3-17　调整字段位置

在标尺 33 位置上单击，创建一个新的分割线，以此调整向导猜测结果。单击“下一步”按钮。

在后面的对话框中设置各个字段的名称和数据类型，如图 3-18 所示。

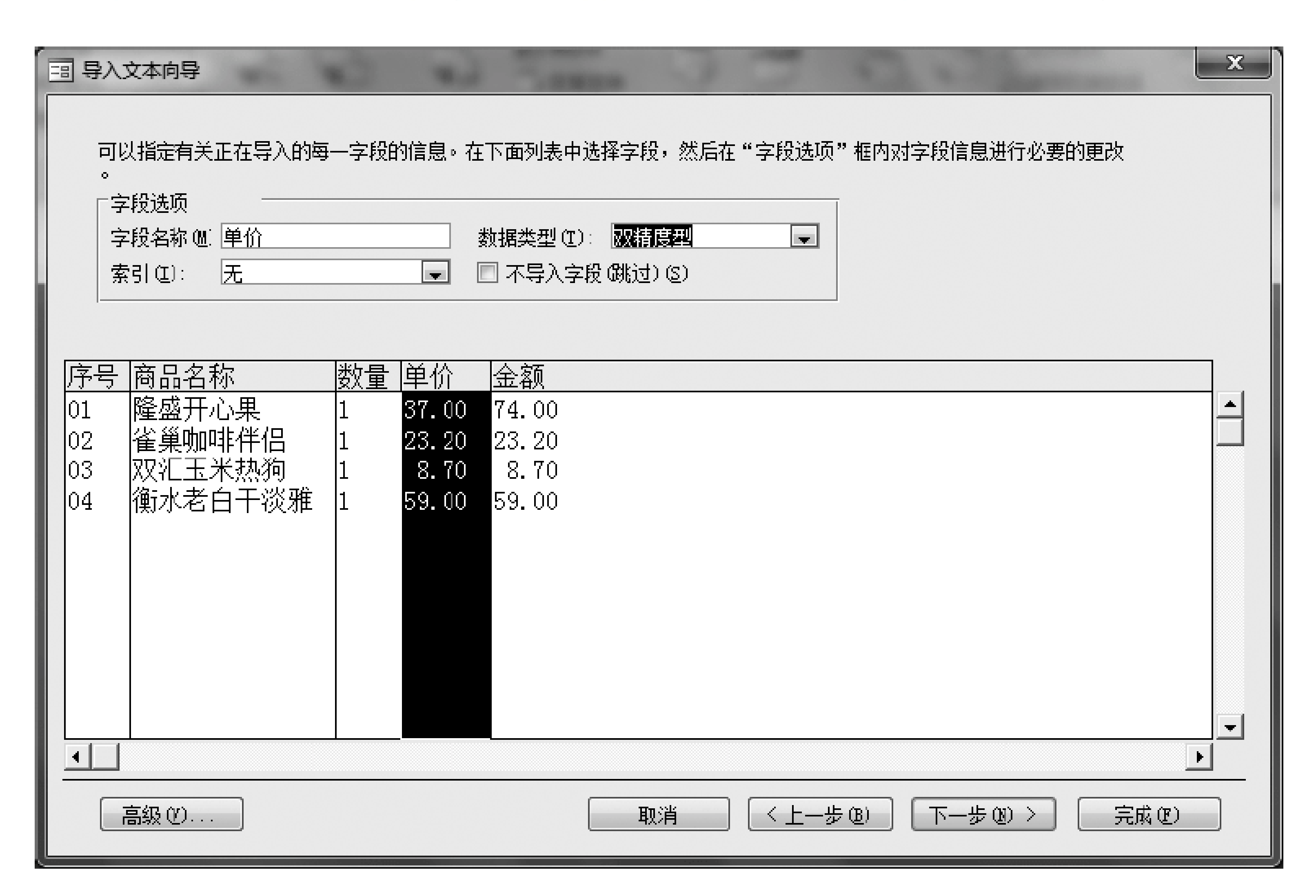


图 3-18　设置数据类型

选择“序号”作为主键，如图 3-19 所示，然后单击“下一步”按钮。

设置目标表的名字为“固定宽度”，以完成导入。

指定表名，单击“完成”按钮，超市.mdb 数据库中就出现了指定名字的新表。然后就可以从 Access 数据库把该表导入到 SQL Server 数据库中了。

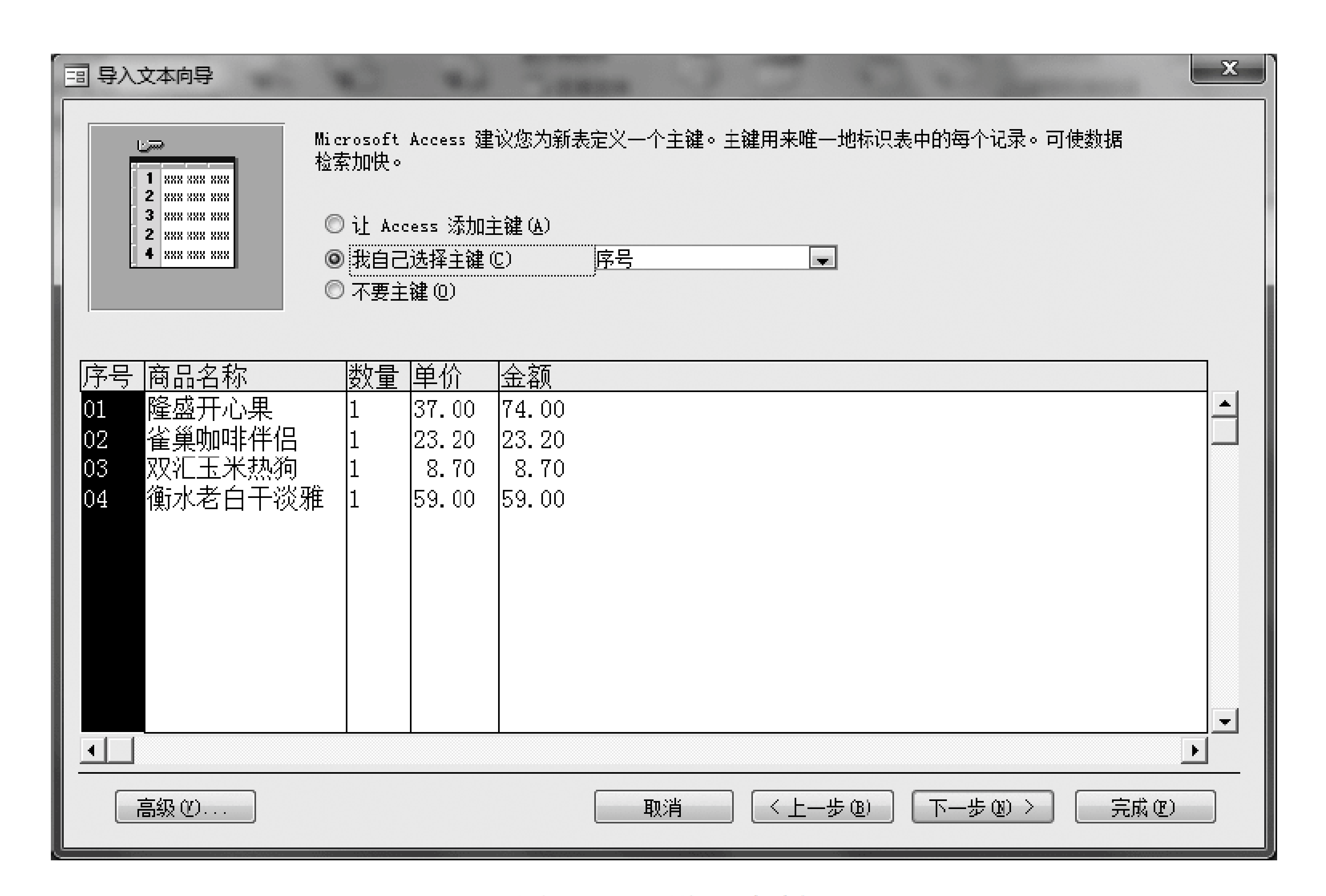


图 3-19　设置主键

注意：在导入 Access 数据库时，文本文件中不要有标题行。

3.3.3　Visual FoxPro 数据表导入 SQL Server

假设在 D：\Data\DBF 文件夹下有 VFP 数据库文件 DBCKJ. DBC，下面将其导入到 SQL Server 中的 TEST 数据库中。

首先下载安装 Microsoft OLE DB Provider for Visual FoxPro，然后在 TEST 数据库上右击，启动导入导出向导。

在"选择数据源"对话框中选择"Microsoft OLE DB Provider for Visual FoxPro"，然后单击"属性"按钮进行设置；在"数据链接属性"对话框中选择或者输入 VFP 数据库的名字；单击"…"按钮，首先会弹出"配置连接"对话框，询问数据库类型：是 VFP 数据库，还是自由表。这里是默认选项，单击"Browse…"按钮选择设置该数据库；保持"选择目标"对话框中的默认设置，其余步骤同样保持默认设置。

注意：VFP 中有两种类型的表：一种是自由表；另一种是数据库中的数据表。自由表是一个单独的. DBF 文件，不属于任何数据库。对于数据库中的数据表，除了有相应的. DBF 文件外，一般在同一文件夹下还有数据库文件. DBC，文件中记载了该数据库中拥有哪些数据表。当导入 .dbc 文件时，应确保其相应的. DBF 文件也存在于正确的磁盘位置上。



高级查询分析技术

本章介绍 SQL 应用的一些高级内容,包括视图、索引、数据字典、临时表、存储过程、函数、触发器、游标等。这些内容有助于在审计数据分析实践中解决较为复杂的案例。

4.1 视　　图

视图即观察数据的窗口。通过 SQL 语句可以从一个表或者多个表中查询符合条件的数据。命名这个查询并保存到数据库中,就是视图。

首先分析一个案例。假设在"东山县医院财务库"中,审计人员希望以如图 4-1 所示格式观察凭证库中的数据。

	源凭证号	科目编码	科目名称	摘要	借方金额	贷方金额
1	2003-10-13-记-5	102	银行存款	收社保中心退...	3064.6000	.0000
2	2003-10-13-记-5	20720	其他	收社保中心退...	.0000	3064.6000
3	2003-10-14-记-6	102	银行存款	交城建税	.0000	554.5800
4	2003-10-14-记-6	102	银行存款	交教育附加	.0000	237.6800
5	2003-10-14-记-6	20701	代扣个人所得税	交个人所得税	84760.0200	.0000
6	2003-10-14-记-6	21003	应交城建税	交城建税	554.5800	.0000
7	2003-10-14-记-6	20703	教育费附加	交教育附加	237.6800	.0000
8	2003-10-14-记-6	102	银行存款	交个人所得税	.0000	84760.0200
9	2003-10-14-记-6	102	银行存款	交营业税	.0000	7922.5000
10	2003-10-14-记-6	21002	应交营业税	交营业税	7922.5000	.0000
11	2003-10-14-记-7	101	现金	于小宝报子女...	.0000	1846.2400
12	2003-10-14-记-7	30301	职工福利基金	于小宝报子女...	1846.2400	.0000

图 4-1　查询凭证库

其特点是:结果集中只有凭证库表中的部分列,列的顺序与凭证库表中列的原始顺序不同,而且只显示 2003 年 10 月份的凭证。那么,通常的思路是在 SQL 查询编辑器中执行查询:

```
SELECT 源凭证号,科目编码,科目名称,摘要,借方金额,贷方金额
FROM 凭证库
WHERE 会计年份=2003 AND 会计月份 =10
```

由于一个审计项目可能持续几个月,每次打开数据库发出这样的查询显然很烦琐。对于这种情况,有的审计人员把查询结果保存在表中。

```
SELECT 源凭证号,科目编码,科目名称,摘要,借方金额,贷方金额
INTO  XX表
```

```
FROM 凭证库
WHERE 会计年份=2003 AND 会计月份 =10
```

以后查询时，只使用以下查询就可以了。

```
SELECT *
FROM XX表
```

这种办法的缺点是数据冗余。可以视图的形式在数据库中保存查询，需要时从视图中进行查询，而不是直接从表中进行查询，以避免数据冗余。使用视图前必须先定义视图。定义视图的 SQL 语句为

```
CREATE VIEW 凭证库借贷视图
AS
SELECT 源凭证号,科目编码,科目名称,摘要,借方金额,贷方金额
FROM dbo.凭证库
WHERE 会计年份=2003 AND 会计月份 =10
```

在对象浏览器中展开"东山县医院财务库"数据库，并展开视图，发现其中有了"dbo.凭证库借贷视图"对象，如图 4-2 所示。



图 4-2　创建及查看视图

定义视图以后，就可以像使用表一样使用视图了。例如：

```
SELECT *
FROM 凭证库借贷视图
```

也可以在查询中增加其他子句：

```
SELECT *
FROM 凭证库借贷视图
ORDER BY 源凭证号
```

可以把视图理解为“虚表”,或者观察表的“窗口”。凭证库借贷视图和凭证库表的关系如图 4-3 所示。

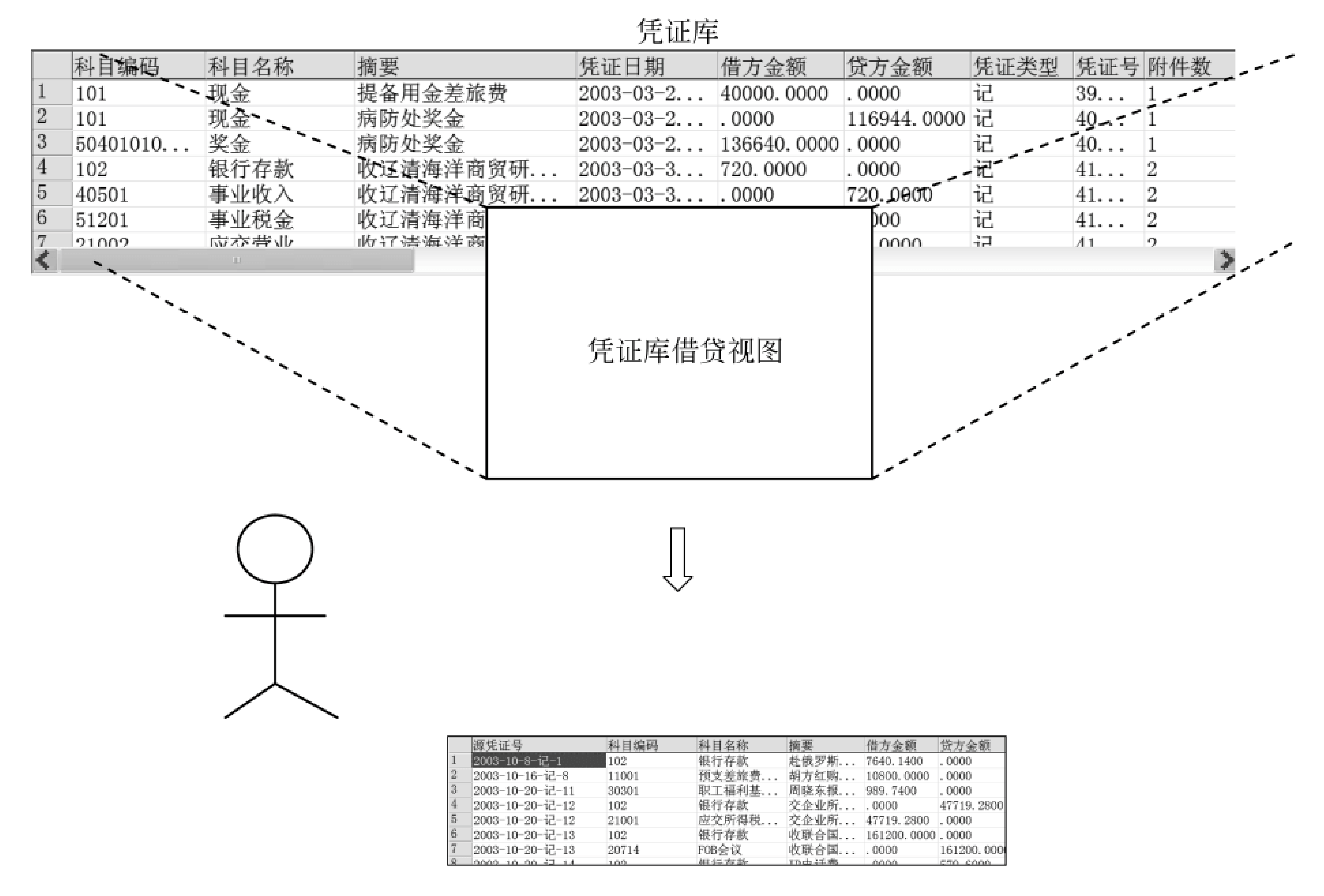


图 4-3 凭证库借贷视图和凭证库表的关系

注意:*视图名中不能含有空格等特殊符号;一般发出创建命令后需刷新对象浏览器,才能在其中显示出来。*

在审计实践中通常对视图进行修改,如对上面的视图增加一列:附件数。那么,首先在查询编辑器对象浏览器中把凭证库借贷视图删除,在 SQL 查询编辑器的对象浏览器中展开视图文件夹,在准备删除的凭证库借贷视图对象上右击,选择“删除”命令。

或者使用命令 DROP VIEW 删除视图。例如:

```
DROP VIEW 凭证库借贷视图
```

然后执行下面的查询重新创建:

```
CREATE VIEW 凭证库借贷视图
AS
SELECT 源凭证号,科目编码,科目名称,摘要,借方金额,贷方金额,附件数
FROM dbo.凭证库
WHERE 会计年份=2003 AND 会计月份 =10
```

如果不先删除旧的视图,而直接创建与旧视图相同名字的视图,就会出现以下错误信息:

```
服务器:消息 2714,级别 16,状态 5,过程 凭证库借贷视图,行 3
```

数据库中已存在名为 '凭证库借贷视图' 的对象。

由于视图是基于“基表”的一个数据观察窗口，并没有对原始数据做任何改变，所以维护了数据的原始性。可以通过视图授权普通用户访问账单表的局部信息。例如，会计科目表的收入类科目的凭证不允许普通用户访问，可以通过为普通用户定义视图实现。

例如，为了对普通用户隐藏药品的进价，可基于“药品”表创建一个没有“单价”列的视图。

```
CREATE VIEW 药品信息
AS
SELECT 药品编号，名称，规格，基本单位，出售单位，因子
FROM 药品
```

然后通过授权操作，使得普通用户可以查看该视图，但不能查看药品表。

使用视图可以方便地实现数据单位变换。例如，假设药品表的进价单位是美元，可创建以人民币元为单位的凭证库。假设汇率为 6.3。

```
CREATE VIEW 药品进价信息(编号，名称，人民币进价)
AS
SELECT 药品编号，名称，进价 * 6.3
FROM 药品
```

使用视图还可以简化复杂查询的构造。对于复杂查询，特别是连接查询，可先建立视图，再基于视图查询。

例如，使用东山县医院业务数据库，以门诊流水号、部门号、部门名称、收费日期、医生号、医生姓名为查询结果集中的列观察数据，可创建“门诊收费详细信息”视图，该视图汇集了来自门诊收费、部门和医生 3 个表的信息，然后从该视图中查询数据。

首先，创建视图。创建视图前，首先清楚从哪些表中获得所期望的列。门诊流水号、部门号、医生号、收费日期来自门诊收费表；而门诊收费表中的部门号和医生号分别引用了部门表中相应的行和医生表中相应的行，根据该引用关系，从部门表中可以找到相应的部门名称，从医生表中可以找到相应的医生姓名。所以，根据该引用关系建立连接条件如下。

```
SELECT
FROM 门诊收费 INNER JOIN 医生 ON 医生.医生号 =门诊收费.医生号
INNER JOIN 部门 ON 部门.部门号 =门诊收费.部门号
```

然后分别从相应表中指定结果集中期望显示的列。首先是来自门诊收费中的列：

```
SELECT 门诊收费.门诊流水号，门诊收费.部门号，门诊收费.收费日期，门诊收费.医生号
FROM 门诊收费 INNER JOIN 医生 ON 医生.医生号 =门诊收费.医生号
INNER JOIN 部门 ON 部门.部门号 =门诊收费.部门号
```

之后，在 SELECT 子句中添加来自部门表的部门名称。注意，位置与要求须保持一致。

```
SELECT 门诊收费.门诊流水号，门诊收费.部门号，部门.名称，门诊收费.收费日期，门诊收费.医生号
```

```
FROM 门诊收费 INNER JOIN 医生 ON 医生.医生号 =门诊收费.医生号
INNER JOIN 部门 ON 部门.部门号 =门诊收费.部门号
```

最后,在 SELECT 子句中添加来自医生表的医生姓名。注意,位置与要求要保持一致。

```
SELECT  门诊收费.门诊流水号, 门诊收费.部门号, 部门.名称,门诊收费.收费日期, 门诊收费.医生号, 医生.姓名
FROM 门诊收费 INNER JOIN 医生 ON 医生.医生号 =门诊收费.医生号
INNER JOIN 部门 ON 部门.部门号 =门诊收费.部门号
```

当看到该查询的执行结果是所期望的结果后,在 SELECT 子句上方添加两行,创建 VIEW。

```
CREATE VIEW 门诊收费详细信息
AS
SELECT 门诊收费.门诊流水号, 门诊收费.部门号, 部门.名称, 门诊收费.收费日期, 门诊收费.医生号, 医生.姓名
FROM 门诊收费 INNER JOIN 医生 ON 医生.医生号 =门诊收费.医生号
INNER JOIN 部门 ON 部门.部门号 =门诊收费.部门号
```

以后就可以基于该视图做简单查询了。

```
SELECT *
FROM 门诊收费详细信息
```

当然,也可以从中查询部分列。

```
SELECT 门诊流水号, 名称, 姓名
FROM 门诊收费详细信息
```

4.2 应用索引加快查询

想象一个庞大的图书馆,如果想从其中取一本书,有两种方法:顺序查找和索引查找。顺序查找指从书架上一本书一本书挨个翻;而索引查找则需要辅助设施——索引卡,先根据书名到索引卡中检索书在书架上的位置,然后直接到书架位置上取书。

从数据库中检索一条记录就相当于从图书馆中找一本书,要想迅速找到该记录,就必须事先建立索引。

4.2.1 索引的类型

1. 从实现角度划分

从实现角度划分,有两种类型的索引:聚簇索引和非聚簇索引。

在聚簇索引中,行的索引顺序与物理存储顺序完全相同。由于聚簇索引的顺序与数据行存放的物理顺序相同,所以聚簇索引最适合范围搜索。因为找到一个范围内开始的

```
FROM 门诊收费 INNER JOIN 医生 ON 医生.医生号 =门诊收费.医生号
INNER JOIN 部门 ON 部门.部门号 =门诊收费.部门号
```

最后,在 SELECT 子句中添加来自医生表的医生姓名。注意,位置与要求要保持一致。

```
SELECT  门诊收费.门诊流水号, 门诊收费.部门号, 部门.名称,门诊收费.收费日期, 门诊收费.医
生号, 医生.姓名
FROM 门诊收费 INNER JOIN 医生 ON 医生.医生号 =门诊收费.医生号
INNER JOIN 部门 ON 部门.部门号 =门诊收费.部门号
```

当看到该查询的执行结果是所期望的结果后,在 SELECT 子句上方添加两行,创建VIEW。

```
CREATE VIEW 门诊收费详细信息
AS
SELECT 门诊收费.门诊流水号, 门诊收费.部门号, 部门.名称, 门诊收费.收费日期, 门诊收费.医生
号, 医生.姓名
FROM 门诊收费 INNER JOIN 医生 ON 医生.医生号 =门诊收费.医生号
INNER JOIN 部门 ON 部门.部门号 =门诊收费.部门号
```

以后就可以基于该视图做简单查询了。

```
SELECT *
FROM 门诊收费详细信息
```

当然,也可以从中查询部分列。

```
SELECT 门诊流水号, 名称, 姓名
FROM 门诊收费详细信息
```

4.2 应用索引加快查询

想象一个庞大的图书馆,如果想从其中取一本书,有两种方法:顺序查找和索引查找。顺序查找指从书架上一本书一本书挨个翻;而索引查找则需要辅助设施——索引卡,先根据书名到索引卡中检索书在书架上的位置,然后直接到书架位置上取书。

从数据库中检索一条记录就相当于从图书馆中找一本书,要想迅速找到该记录,就必须事先建立索引。

4.2.1 索引的类型

1. 从实现角度划分

从实现角度划分,有两种类型的索引:聚簇索引和非聚簇索引。

在聚簇索引中,行的索引顺序与物理存储顺序完全相同。由于聚簇索引的顺序与数据行存放的物理顺序相同,所以聚簇索引最适合范围搜索。因为找到一个范围内开始的

行后，可以很快地取出后面的行。

因为一个表中的行只能有一个物理顺序，所以每个表只能有一个聚簇索引。如果表中没有创建其他的聚簇索引，则在表的主键列上自动创建聚簇索引。

非聚簇索引并不在物理上排列数据，即索引中的逻辑顺序并不等同于表中行的物理顺序。索引仅记录指向表中行的位置信息，通过这些信息可以在表中快速地定位数据。可以为表中每个常用于查询的列定义非聚簇索引。

非聚簇索引的特点使它很适合于那种直接匹配单个条件的查询，而不太适合于返回大量结果的查询。为一个表建立的索引默认都是非聚簇索引。如果在某一列上设置唯一性约束，也会自动在该列上创建非聚簇索引。

2 从功能角度划分

从功能角度划分，也有两种索引：唯一性索引（Unique Indexes）和非唯一性索引。

唯一性索引能够保证在创建索引的列或多列的组合上不包括重复的数据，聚簇索引和非聚簇索引都可以是唯一性索引。在创建主键约束和唯一性约束的列上会自动创建唯一性索引。

创建唯一性索引时，应保证创建索引的列不包括重复的数据，并且没有两个或两个以上的空值。因为创建索引时将两个空值也视为重复的数据，如果有这种数据，必须先将其删除，否则索引不能成功创建。

4.2.2 索引的创建

可以在 SQL 编辑器中使用 SQL 命令创建索引。例如，在“门诊收费明细”表的“项目编号”列上建立索引 IX_MZ_XMBH。

```
CREATE INDEX IX_MZ_XMBH
ON 门诊收费明细 (项目编号)
```

创建索引后，就为实现对大规模表的快速检索提供了基础。注意：①对于小表，索引对于提高检索性能没有帮助；②索引提高了数据的检索速度，但降低了数据的更新速度；③对于用来进行 JOIN 的列，建立索引将大大提升连接性能；④不要对经常更新的列建立索引。

4.3 数据字典

数据字典是关于数据的数据，可以通过数据字典获得全面的数据库信息。数据字典也是以数据表和视图为主要存在形式的。

使用 SQL Server 时，需要事先了解关于数据的信息，如数据库对应的磁盘文件，数据库中的表和视图，某个表或者视图中列的个数以及每一列的名称、数据类型、长度、精度（有效数字的位数）、描述，表上定义的约束（NOT NULL、RIMARY KEY、FOREIGN KEY、UNIQUE、CHECK），表上定义的索引、触发器。

下面以“东山县医院业务库”为例，介绍如何在管理器中查看数据字典信息。

4.3.1 数据文件和事务日志文件

在对象资源管理器中的“东山县医院业务库”上右击，选择“属性”，然后单击“文件”选项页，可以看到数据库“东山县医院业务库”的数据库文件的文件名是“D:\东山县医院业务库_Data.MDF”，逻辑文件名是“东山县医院业务库_Data”。物理文件名是被操作系统使用的文件名；逻辑文件名是在数据库服务器中使用的数据文件名。路径指该文件所在的文件夹。“东山县医院业务库”的事务日志文件的文件名是“D:\东山县医院库_Log.LDF”；逻辑文件名是“东山县医院业务库_Log”。

4.3.2 表定义

如果想查看“东山县医院业务库”数据库中有哪些表，各表是如何定义的，表上有哪些约束和索引，那么在对象资源管理器中单击“东山县医院业务库”左侧的“+”，展开该结点，然后单击“表”，可以看到类型为“用户”的表有8个，分别是门诊收费明细、门诊收费、部门、药品、检查项目、医生、住院收费明细和住院收费。被审单位的原始生产数据存在于用户表中。其他表是SQL Server自动创建的表，存放数据字典信息。

单击“医生”表前的“+”，展开“医生”表，可以看到“医生”表中有医生号、姓名和部门号3列及各列的数据类型等。部门号列上的取值允许为空，其他列上的取值不允许为空。

在管理器“键”项目组定义的主键约束的名字是PK_医生，该主键建立在“医生号”列上。把“医生号”定义为主键，使得数据库能够自动维护在整个表的所有行在“医生号”上的值不重复而且不为空。还定义了外键约束，其名字为“FK_医生_部门”，建立在“部门号”列上。

4.4 临 时 表

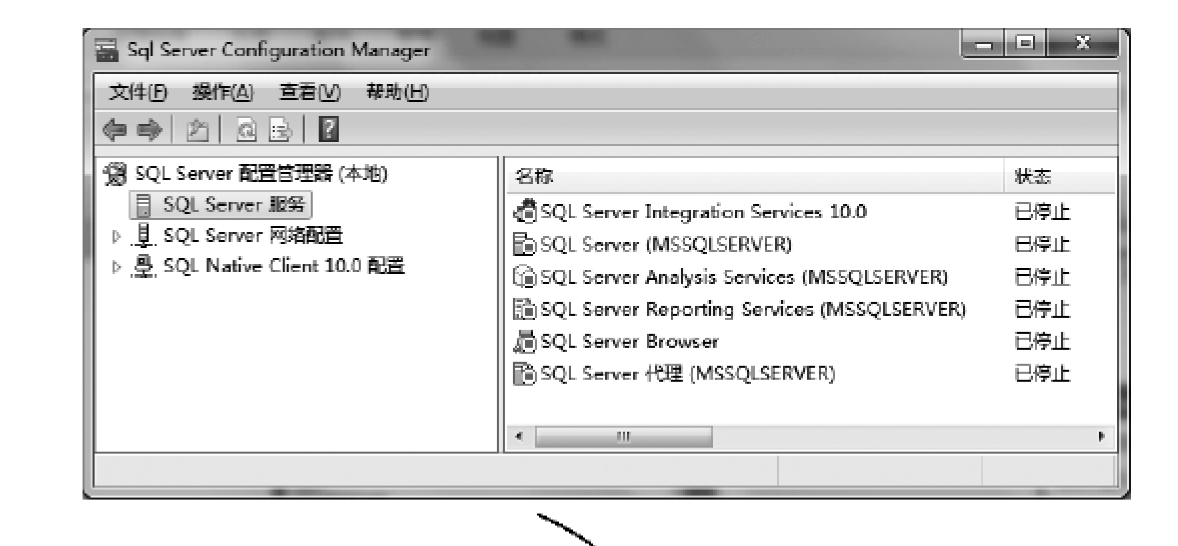
4.4.1 客户与数据库服务器的连接

SQL Server运行在服务器模式下。客户(如SQL查询编辑器)接受用户的输入，并将用户输入通过与服务器的连接发送给SQL Server，如图4-4所示。

每一个查询窗口都与服务器保持一个连接。可通过SQL查询编辑器的状态栏查看当前的连接数目。

默认情况下，SQL Server允许无限制连接数目(默认值0)。在对象资源管理器中右击服务器，选择“属性”，在SQL Server属性对话框的“连接”选项卡中可以重新设置连接数目，如改为6。重新启动SQL Server服务，使得该设置有效。那么，当在SQL查询编辑器中打开第7个查询窗口时，将会出现失败的提示信息。

客户端和服务器可以部署在不同计算机上。假设所要访问的实例为SQLDEV，若要远程登录SQL Server，则应满足两个前提：一是SQL Server的登录身份为Network Service；二是服务器启用了Named Pipes协议和TCP/IP。



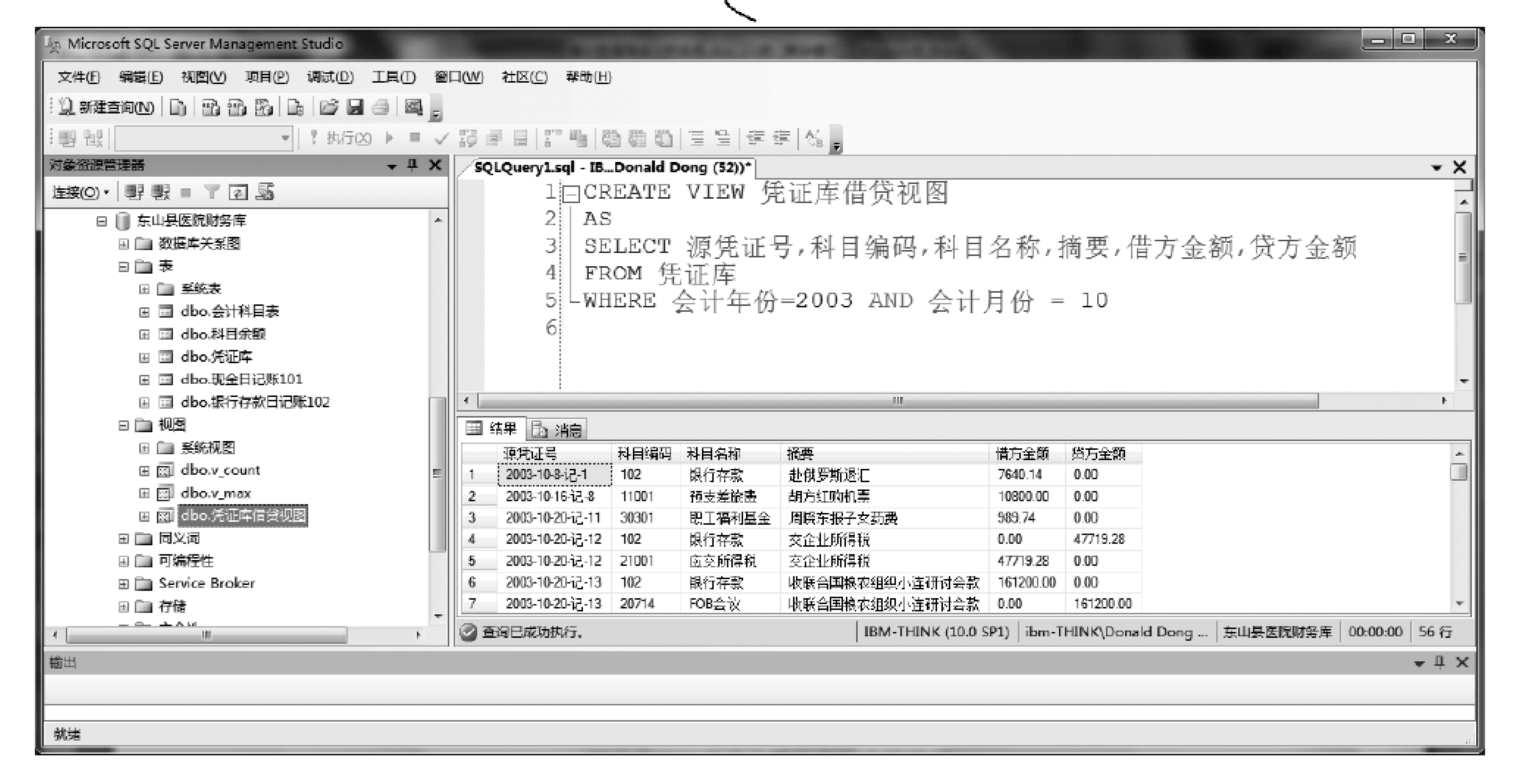


图 4-4　连接

选择启动菜单|Microsoft SQL Server 2008 R2|配置工具|SQL Server 配置管理器(本地)，出现如图 4-5 所示的对话框。

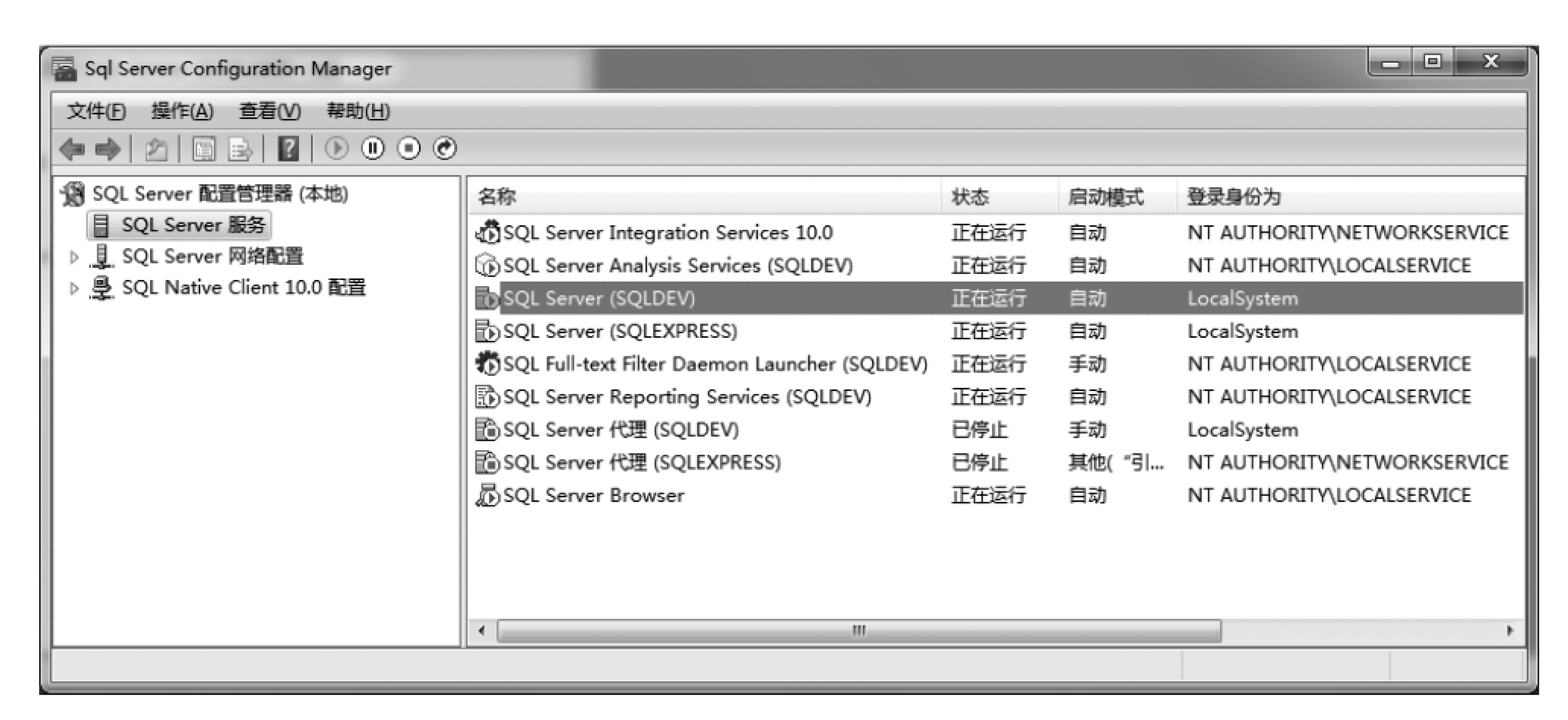


图 4-5　配置管理器

从图 4-5 中观察到 SQL Server(SQLDEV)的登录身份为 LocalSystem。右击该行，从弹出的菜单中选择“属性”，出现“SQL Server(SQLDEV)属性”对话框，在该对话框中把“内置账户”由 Local System 改为 Network Service，如图 4-6 所示。



图 4-6 “SQL Server(SQLDEV)属性”对话框

再单击服务器的 SQL Server 网络配置，双击“SQLDEV 的协议”，启用其中的 Named Pipes 和 TCP/IP，如图 4-7 所示。



图 4-7 启用协议

通过重新启动服务使配置生效。

4.4.2 临时表的创建与删除

临时表在数据库中临时存在，当用户从数据库中注销或者连接中断后，临时表被自动删除。临时表位于 tempdb 数据库中，tempdb 数据库是 SQL Server 安装时自动创建的。

有本地和全局两种类型的临时表，二者在名称、可见性和可用性上均不相同。本地临时表的名称以“#”开头；它们仅对当前的用户连接是可见的；当用户从 SQL Server 实例断开连接时被自动删除。全局临时表的名称以“##”开头，创建后对任何用户都是可见

的，当所有引用该表的用户从 SQL Server 断开连接时被自动删除。例如，如果用户甲创建名为"#劳务费"的本地临时表，只有甲能对该表执行操作且在断开连接时该表被自动删除。如果创建名为"##劳务费"的全局临时表，数据库中的任何用户均可对该表执行操作。如果该表在甲创建后没有其他用户使用，则当甲断开连接时该表被自动删除；如果该表在甲创建后有其他用户使用，则 SQL Server 在所有用户断开连接后删除该表。

创建临时表的语法如下。

```
CREATE TABLE #<表名>  (<列名 1><数据类型>,…,<列名 n><数据类型>)
```

该命令在 tempdb 中创建临时表，而且该表仅对其创建者可见，当其创建者从数据库中断开连接后，被自动删除。

创建全局临时表的语法如下。

```
CREATE TABLE ##<表名>  (<列名 1><数据类型>,…,<列名 n><数据类型>)
```

下面的操作序列演示了临时表的特点。用户甲创建了临时表#T，然后向该表中插入了一行数据；用户乙连接数据库，会发现无法使用该表；用户甲断开连接，然后重新连接，会发现该临时表自动消失了。

步骤 1，从 SQL 查询编辑器的"文件"菜单中选择"连接"，用户甲登录；创建临时表。

```
CREATE TABLE  #T (A char(30), B int)
INSERT  #T  VALUES ("YYY", 1)
```

步骤 2，从 SQL 查询编辑器的"文件"菜单中选择"连接"，用户乙登录。

步骤 3，在用户乙的连接中：

```
SELECT *  FROM  #T
```

显示"无此对象"的出错信息。

步骤 4，从 SQL 查询编辑器的"文件"菜单中选择"断开"，断开甲的连接。

步骤 5，从 SQL 查询编辑器的"文件"菜单中选择"连接"，甲重新登录。

步骤 6，发出以下命令：

```
SELECT *  FROM  #T
```

显示"无此对象"的出错信息。

也可以使用 INTO 子句创建临时表。例如：

```
SELECT *  INTO #T FROM 凭证库
```

4.5 设计脚本完成计算

脚本(script)是由数据库服务器引擎执行的 T-SQL 语句序列。

4.5.1 案例：计算个人所得税

假设税法规定，个人月收入 3500 元以内，个人所得税为 0；个人月收入超过 3500 元，

个人所得税为超出部分乘以 0.05。

现要求给定个人月收入，编程计算其应交纳的所得税。

首先安排一个存储单元存放给定的个人月收入，然后安排一个存储单元存放计算结果，并将这两个存储单元分别命名为@S 和@T，然后通过 T-SQL 脚本进行计算。完成计算的 T-SQL 脚本如下。

```
DECLARE @S AS DECIMAL(7,2)
DECLARE @T AS DECIMAL(7,2)
SET @S = 1200
IF @S<= 3500    SET @T = 0
IF @S > 3500 SET @T = (@S - 3500) * 0.05
PRINT @T
```

当查询编辑器看到

```
DECLARE @S AS DECIMAL(7,2)
DECLARE @T AS DECIMAL(7,2)
```

后，就安排了两个存储单元：



准备存放带小数点的数值数据。

语句 SET @S = 1200 的含义是把数值 1200 存放到@S 中：



语句

```
IF @S<= 3500    SET @T = 0
```

的含义是如果@S 中的数值小于或等于 3500，就把 0 存放到@T 中。

语句

```
IF @S > 3500 SET    @T = (@S - 3500) * 0.05
```

的含义为如果@S 中的数值大于 3500，就把表达式(@S－3500) * 0.05 的计算结果存放到存储单元@T 中。

因为当前@S 中存放的是 1200，而 1200<=3500，所以应当把 0 存放到@T 中。



程序完成计算后，计算的结果存放在存储单元中，并不能自动显示在屏幕上。命令 PRINT @T 的作用是把@T 中的数据显示在屏幕上。

可以看到，脚本是一个语句序列，该序列完成一定的功能。脚本中需要声明一些存储单元用来存放被程序处理的数据，并使用一个名字进行标识，称为变量，如@S，@T。向存储单元中存放数据称为赋值，如 SET @S = 1200 的含义是把数据 1200 存放到以@S

标识的存储单元中。脚本是顺序执行的，即按照语句的书写顺序逐条执行。

4.5.2 标识符、语句和注释

标识符是由字母或者数字字符组成的字符串。通常用它们表示服务器名、数据库名、数据库对象名、常量、变量、存储过程名称等。标识符的命名须遵守以下规则：标识符最多有 128 个字符；标识符的第一个字符必须为字母、下画线、@或者＃。以@或者＃开头的标识符具有特殊意义。程序中使用的变量要求是以@为首的标识符。

一条 T-SQL 语句至少需要包含一个命令，即一个表明动作意义的动词。例如，SELECT 动词请求服务器检索出若干数据行；UPDATE 动词要求服务器改变特定行的内容。

T-SQL 脚本包含一个或多个语句。

语句 GO 的意思是将上一 GO 命令后输入的所有语句发送给 SQL Server。

注释是程序代码中不执行的文本字符串(也称为注解)。注释可用于说明代码或暂时禁用正在进行诊断的部分 T-SQL 语句和批处理。使用注释对代码进行说明，可使程序代码更易于维护。注释可用于说明程序名称、作者姓名和主要代码更改的日期，称为序言性注释；也可用于描述算法，称为功能性注释。

T-SQL 支持两种注释字符：

--(双连字符)。这些注释字符可与要执行的代码处在同一行，也可另起一行。从双连字符开始到行尾均为注释。对于多行注释，必须在每个注释行的开始使用双连字符。

/* ... */(正斜杠-星号对)。这些注释字符可与要执行的代码处在同一行，也可另起一行，甚至在可执行代码内。从开始注释对 (/*) 到结束注释对 (*/) 之间的内容为注释部分。对于多行注释，必须使用开始注释字符对 (/*) 开始注释，使用结束注释字符对 (*/) 结束注释。注释行上不应出现其他注释字符。下面是示例。

```
--多行注释的第一行
--多行注释的第二行
  SELECT * FROM 药品
/*多行注释的第一行
多行注释的第二行 */
  SELECT * FROM 医生
```

4.5.3 变量

T-SQL 变量是保存特定类型的单个数据值的对象。变量必须先声明，后使用。

用 DECLARE 语句声明变量，并用 SET 或 SELECT 语句给其指派值。所有变量在声明后都被初始化为 NULL。

声明变量的语法：

```
DECLARE  @局部变量名 数据类型
```

例如：

```
DECLARE @S INT
```

注意，变量名必须以字符“@”开头。

变量的赋值语法为

```
SET @局部变量名 =<值>|<表达式>
```

声明变量后，所有变量均初始化为 NULL。使用 SET 语句将一个不是 NULL 的值赋给变量。当初始化多个变量时，为每个局部变量使用一个单独的 SET 语句。

例如，计算两个变量的和并显示结果的脚本为

```
DECLARE @x int, @y int, @r int
SET @x = 156
SET @y = 26
SET @r = @x + @y
PRINT @r
```

4.5.4 流控制语句 IF-ELSE

在下面计算个人所得税的例子中：

```
DECLARE @S AS DECIMAL(7,2)
DECLARE @T AS DECIMAL(7,2)
SET @S = 1200
IF @S<= 3500   SET @T = 0
IF @S > 3500 SET @T = (@S - 3500) * 0.05
PRINT @T
```

计算部分是 IF 开头的语句。这样的语句只有在 IF 后面的条件满足时，才执行后面的命令，称为控制流语句。

IF-ELSE 是典型的分支控制结构，其用法有两种：有 IF 而无 ELSE 的 IF 语句和有 IF 也有 ELSE 的 IF 语句。

有 IF 而无 ELSE 的 IF 语句的语法如下。

```
IF <条件>
  <语句|语句块 1>
```

当＜条件＞成立时，执行 IF 语句中的语句或语句块；当条件不成立时，跳过 IF 语句后的语句或语句块。

4.5.5 BEGIN…END

BEGIN 和 END 语句用于将多个 T-SQL 语句组合为一个语句块。任何时候，当控制流语句必须执行一个包含两条或两条以上 T-SQL 语句的语句块时，使用 BEGIN 和 END 语句。

例如，计算个税，如果收入小于或等于 3500 元，个税为 0，并显示“低收入”；如果收入

大于 3500 元，税率为 5%，并显示“工薪收入”。一种可能的脚本设计为

```
DECLARE @income int, @tax int
SET @income = 1000
IF @income <= 3500
    SET @tax = 0
    PRINT '低收入' --当条件满足,显示“低收入”
IF @income > 3500
    SET @tax = (@income- 3500) * 0.05
    PRINT '工薪收入'
PRINT @tax
```

则当@income=1000 时会出现：

```
低收入
工薪收入
0
```

的输出结果。这与题目要求不一致。如果想让语句

```
SET @tax = 0
PRINT '低收入' --当条件满足,显示“低收入”
```

作为一个整体，在条件成立的时候执行，则会用到 BENGIN…END。用 BEGIN 和 END 语句后的效果如下。

```
DECLARE @income int, @tax int
SET @income = 8000
IF @income <= 3500
  BEGIN
    SET @tax = 0
    PRINT '低收入' --当条件满足,显示“低收入”
  END
IF @income > 3500
  BEGIN
    SET @tax = (@income- 3500) * 0.05
    PRINT '工薪收入'
  END
PRINT @tax
```

注意，BEGIN 和 END 语句必须成对使用。任何一条语句均不能单独使用。BEGIN 语句行后为 T-SQL 语句块，最后，END 语句行指示语句块结束。

4.5.6 IF ELSE 语句

有 IF 也有 ELSE 的 IF 语句的语法如下。

```
IF <条件>
```

```
<语句|语句块 1>
ELSE
<语句|语句块 2>
```

当<条件>成立时，执行 IF 语句后的语句或语句块，然后控制跳到 ELSE 语句后的语句或语句块之后的点；当<条件>不成立时，跳过 IF 语句后的语句或语句块，而执行 ELSE 语句后的语句或语句块。

例如，计算个税，如果收入小于或等于 3500 元，个税为 0，并显示“低收入”；如果收入大于 3500 元，税率为 5%，并显示“工薪收入”。完成此计算的脚本为

```
DECLARE @income int, @tax int
SET @income = 2000
--如果收入小于或等于 3500 元,个税为 0;否则税率为 5%.
IF @income <= 3500
  BEGIN
   SET @tax = 0
    PRINT '低收入'
  END
ELSE
  BEGIN
    SET @tax = (@income- 3500) * 0.05
    PRINT '工薪收入'
  END
PRINT @tax
```

在此脚本中，IF 和 ELSE 对齐，其各自控制的语句缩进并对齐。

再如，根据数值判断借贷方向：1 表示“借”；－1 表示“贷”；0 表示“平”；其他数值表示“未知”。完成此计算的脚本为

```
DECLARE @f  int
SET @f = 1
IF @f = 1
    PRINT '借'
ELSE
    IF @f = -1
        PRINT '贷'
    ELSE
        IF @f = 0
               PRINT '平'
        ELSE
               PRINT '未知'
```

4.5.7　CASE 表达式

CASE 具有两种格式：CASE 简单表达式和 CASE 搜索表达式。CASE 简单表达式

将某个表达式与一组简单表达式进行比较，以确定结果；CASE 搜索表达式计算一组布尔表达式，以确定结果。

（1）CASE 简单表达式的语法如下。

```
CASE <测试表达式>
     WHEN <表达式 1> THEN <结果表达式 1>
     WHEN <表达式 2> THEN <结果表达式 2>
      ⋮
     WHEN <表达式 n> THEN <结果表达式 n>
     [ELSE <结果表达式 n+1> ]
END
```

执行时，先计算测试表达式，然后按从上到下的顺序对每个 WHEN 子句的表达式进行计算。如果某个 WHEN 子句的表达式值与测试表达式的值相匹配，则返回第一个相匹配的 WHEN 子句的＜结果表达式＞的值。如果没有匹配，则当指定 ELSE 子句时，SQL Server 将返回＜结果表达式 n+1＞的值；若没有指定 ELSE 子句，则返回 NULL 值。

例如，把数值表示的借贷方向显示为汉字：1 显示为“借”；－1 显示为“贷”；0 显示为“平”；其他数值显示为“错误”。如果使用 IF 语句，则脚本为

```
DECLARE @flag int
SET @flag = 2
IF @flag = 1
  PRINT '借'
ELSE
  IF @flag = -1
    PRINT '贷'
  ELSE
    IF @flag = 0
        PRINT '平'
    ELSE
        PRINT '错误'
```

虽然产生了正确的结果，但是这样的程序使人看起来眼花缭乱。T-SQL 提供了更加简洁的表达方式，使得程序更简单、易懂。这就是使用 CASE 表达式：

```
--若是 1,则显示"借";若是-1,则显示"贷";若是 0,则显示"平";其他,显示"错误"
DECLARE @flag int, @r nchar(2)
SET @flag = 2
SET @r= CASE @flag
     WHEN   1 THEN   '借'
     WHEN   -1 THEN   '贷'
     WHEN   0 THEN   '平'
     ELSE '错误'
END
print @r
```

注意，CASE要和END配对出现。

(2) CASE搜索表达式的语法如下。

```
CASE
    WHEN<条件 1>THEN<结果表达式 1>
    WHEN<条件 2>THEN<结果表达式 2>
    ⋮
    WHEN<条件 n>THEN<结果表达式 n>
    [ELSE<结果表达式 n+1>]
END
```

执行时，按从上到下的顺序为每个WHEN子句的条件求值，返回第一个成立的条件对应的＜结果表达式＞的值。如果所有条件都不成立，则当有ELSE子句时，将返回＜结果表达式 n＋1＞；若没有指定ELSE子句，则返回NULL值。

把数值表示的借贷方向转换为汉字的另一个实现脚本如下。

```
DECLARE @flag int, @r char(2)
SET @flag = 0
SET @r= CASE
    WHEN @flag = 1 THEN  '借'
    WHEN @flag = -1 THEN '贷'
    WHEN @flag = 0 THEN  '平'
    ELSE '错误'
END
print @r
```

再看一个例子。空气污染指数(Air Pollution Index，API)是评估空气质量状况的一组数字，它能告诉您今天或明天您呼吸的空气是清洁的，还是受到污染的，以及您应当注意的健康问题。空气污染指数关注的是吸入受到污染的空气以后几小时或几天内人体健康可能受到的影响。空气污染指数划分为0～50、51～100、101～150、151～200、201～300和大于300这6档，对应于空气质量的6个级别，指数越大，级别越高，说明污染越严重，对人体健康的影响也越明显。任意给定一个指数，试设计脚本，计算其属于哪一档。

(1) 使用IF语句的脚本为

```
--输入空气污染指数,输出其所属的级别(1~6)
DECLARE @api int, @level int
SET @api = 30
IF @api >= 0 AND @api <= 50
  SET @level = 1
IF @api >= 51 AND @api <= 100
  SET @level = 2
IF @api >= 101 AND @api <= 150
  SET @level = 3
IF @api >= 151 AND @api <= 200
  SET @level = 4
```

```
IF @api >=201 AND @api <=300
  SET @level =5
IF @api >=301
  SET @level =6
PRINT cast (@level AS nchar(1)) + '级污染'
```

（2）使用 CASE 表达式的脚本为

```
DECLARE @api int, @level int
SET @api =300
SET @level =CASE
  WHEN @api >=0 AND @api <=50 THEN 1
  WHEN @api >=51 AND @api <=100 THEN 2
  WHEN @api >=101 AND @api <=150 THEN 3
  WHEN @api >=151 AND @api <=200 THEN 4
  WHEN @api >=201 AND @api <=300 THEN 5
  WHEN @api >=301  THEN 6
END
PRINT cast (@level AS nchar(1)) + '级污染'
```

下面例子的应用较为广泛。在会计信息系统中，常见的金额结构有借方金额＋贷方金额、借贷方向＋金额、带正负号的金额。审计实践中常把后两种方式转换为第一种。例如，将“借贷方向＋金额”方式的表

源凭证号	科目代码	科目名称	借贷方向	金额/元
2013-3-31-记-41	102	银行存款	借	720
2013-3-31-记-41	40501	事业收入	贷	720
2013-3-31-记-41	51201	事业税金	借	39.6
2013-3-31-记-41	21002	应交营业税	贷	36
2013-3-31-记-41	21003	应交城建税	贷	2.52
2013-3-31-记-41	20703	教育费附加	贷	1.08

转换为

源凭证号	科目代码	科目名称	借方金额/元	贷方金额/元
2013-3-31-记-41	102	银行存款	720	
2013-3-31-记-41	40501	事业收入		720
2013-3-31-记-41	51201	事业税金	39.6	
2013-3-31-记-41	21002	应交营业税		36
2013-3-31-记-41	21003	应交城建税		2.52
2013-3-31-记-41	20703	教育费附加		1.08

可设计脚本为

```
SELECT 源凭证号,科目代码, 科目名称 CASE
                    WHEN 借贷方向='借' then 金额
                    ELSE null
                END AS 借方金额,
            CASE
                    WHEN 借贷方向='贷' then 金额
                    ELSE null
                END AS 贷方金额
INTO 借贷凭证表
FROM 凭证表
```

也可以使用如下脚本。

```
SELECT 凭证号,科目代码,科目名称,借方金额 =CASE
                    WHEN 借贷方向='借' then 金额
                    ELSE null
                END,
            贷方金额 =CASE
                    WHEN 借贷方向='贷' then 金额
                    ELSE null
                END
INTO 借贷凭证表
FROM 凭证表
```

把带正负号的金额结构转换成借贷格式的结构：

源凭证号	科目代码	科目名称	金额/元
2013-3-31-记-41	102	银行存款	720
2013-3-31-记-41	40501	事业收入	－720
2013-3-31-记-41	51201	事业税金	39.6
2013-3-31-记-41	21002	应交营业税	－36
2013-3-31-记-41	21003	应交城建税	－2.52
2013-3-31-记-41	20703	教育费附加	－1.08

可使用脚本：

```
SELECT 凭证号,科目代码, 科目名称, 借方金额 =CASE
                                    WHEN 金额>0 then 金额
                                    ELSE null
                                END,
                            贷方金额 =CASE
                                    WHEN 金额<0 then abs(金额)
```

```
    ELSE null
END
```

4.5.8 WHILE 语句

例如，设计脚本，计算 1+2+3+…+10 的结果。可以考虑声明一个存放结果的存储单元@sum，把各个数逐个累加上去。脚本如下。

```
DECLARE  @sum int
SET @sum = 0
SET @sum = @sum + 1
SET @sum = @sum + 2
SET @sum = @sum + 3
SET @sum = @sum + 4
SET @sum = @sum + 5
SET @sum = @sum + 6
SET @sum = @sum + 7
SET @sum = @sum + 8
SET @sum = @sum + 9
SET @sum = @sum + 10
PRINT @sum
```

图 4-8 展示了随着语句的逐条执行，变量@sum 的内容变化。

SET @sum = 0	0
SET @sum = @sum + 1	1
SET @sum = @sum + 2	3
SET @sum = @sum + 3	6
SET @sum = @sum + 4	10
SET @sum = @sum + 5	15
SET @sum = @sum + 6	21
SET @sum = @sum + 7	28
SET @sum = @sum + 8	36
SET @sum = @sum + 9	45
SET @sum = @sum + 10	55

图 4-8　变量@sum 的内容变化

可以看到,对 1,2,3,…,10 进行加法的 10 条语句非常相似。自然也可以想到,如果加到 100,甚至加到 1000,这样的程序设计将不现实。根据语句的相似性,可以设计如下语句块,让其反复执行:

```
SET @sum = @sum + @i
```

让@i 的值从 1 变化到 10。从其变化规律看,每次@i 增 1 即可。

```
SET @sum = @sum + @i
SET @i = @i + 1
```

但是,注意一个问题:当第一次执行该语句块时,@sum 和@i 的值是多少?所以,在反复执行之前,必须清楚@sum 和@i 的初值。

```
SET @sum = 0
SET @i = 1
```

还有一个问题:重复执行语句块多少次?也就是说,何时停止重复执行?对于本例而言,希望其在@i=10 时停止执行,即在满足条件@i <= 10 时要重复执行。

总而言之,希望这样解决问题:设置@sum 的初值为 0;设置@i 的初值为 1,当满足条件@i <= 10 时,要重复执行如下语句块。

```
SET @sum = @sum + @i
SET @i = @i + 1
```

这种情况可使用 T-SQL 的 WHILE 语句实现。

```
DECLARE @i int, @sum int
SET @sum = 0
SET @i = 1
WHILE @i <= 10
BEGIN
    SET @sum = @sum + @i
    SET @i = @i + 1
END
```

图 4-9 展示了随着语句块的重复执行,变量@sum 和@i 的内容变化过程。

可以再添加语句,将结果显示在屏幕上。

再如,设计脚本,计算 1+3+5…+99 的结果。脚本如下。

```
DECLARE @i int, @sum int
SET @i = 1
SET @sum = 0
WHILE @i <= 99
BEGIN
    SET @sum = @sum + @i
    SET @i = @i + 2
END
```

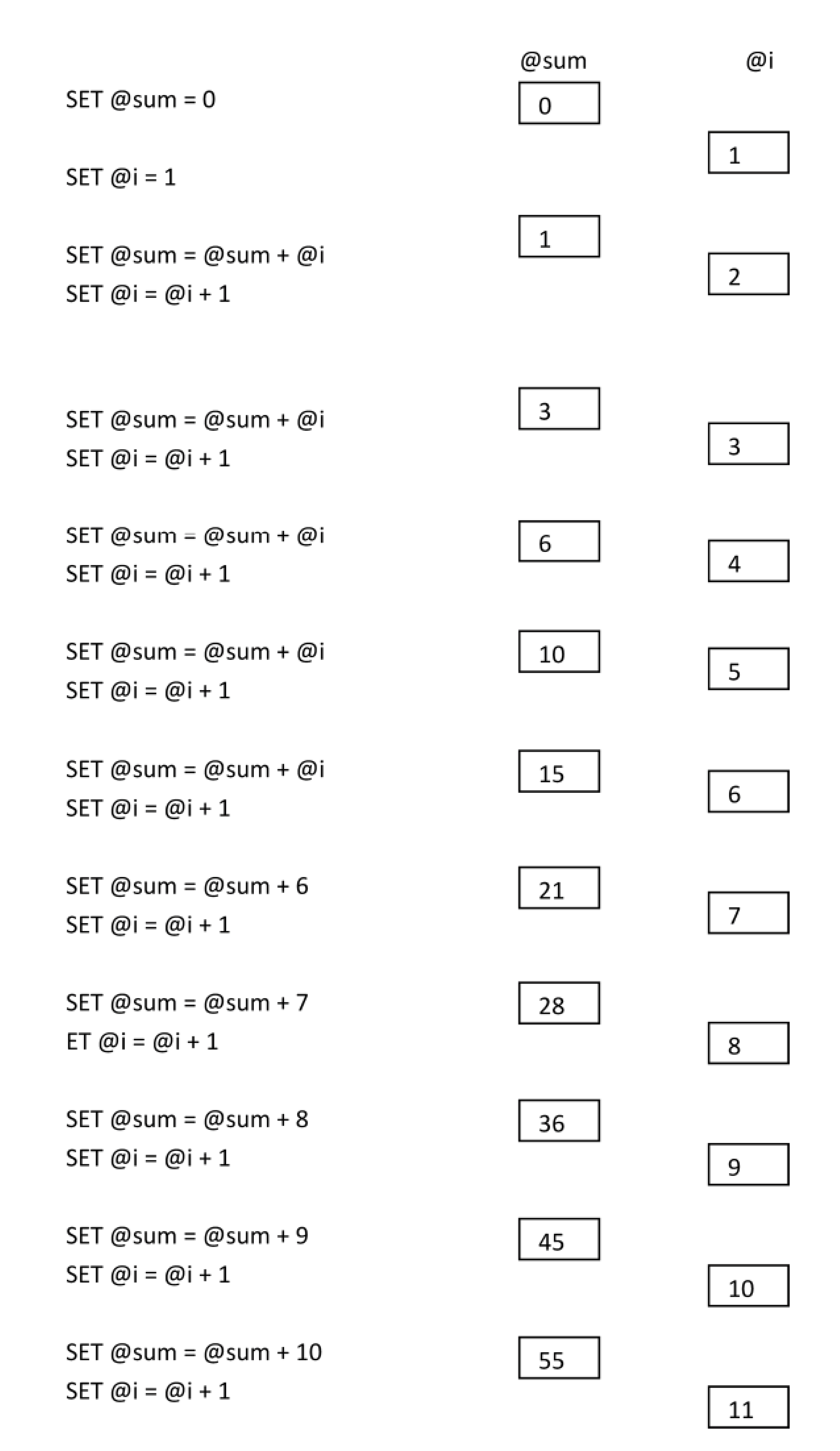


图 4-9　变量@sum 和@i 的内容变化过程

```
PRINT @sum
```

BREAK 或 CONTINUE 语句通常与 WHILE 同时使用。BREAK 语句立即退出其所在的 WHILE 循环，CONTINUE 语句立即执行下次的 WHILE 循环。

例如，下面的脚本使用 BREAK 从 0 累加到 50。

```
DECLARE @i int, @sum int
SET @i = 0
SET @sum = 0
WHILE @i < 100
BEGIN
  SET @i = @i + 1
```

```
    SET @sum = @sum + @i
    IF @i = 50    BREAK
END
PRINT @sum
```

下面的脚本寻找 i 为多大时，累加和开始大于 500。

```
DECLARE @i int, @sum int
SET @i = 0
SET @sum = 0
WHILE @i < 100
BEGIN
    SET @i = @i + 1
    SET @sum = @sum + @i
                            IF @sum> 500
                    BREAK
END
PRINT str(@sum) + ',' + str(@i)
```

下面的脚本实现从 1 累加到 100，但跳过 50。

```
DECLARE @i int, @sum int
SET @i = 0
SET @sum = 0
WHILE @i < 100
BEGIN
        SET @i = @i + 1
                            IF @i = 50
                                    CONTINUE
        SET @sum = @sum + @i
    END
PRINT @sum
```

T-SQL 部分常用控制流关键字见表 4-1。

表 4-1　T-SQL 部分常用控制流关键字

关　键　字	描　　述
IF…ELSE	定义条件以及当条件为 TRUE 或 FALSE 时的操作
BEGIN…END	定义语句块
WHILE	当特定条件为 TRUE 时重复执行循环体中的语句
BREAK	立即退出所在层的 WHILE 循环
CONTINUE	跳过循环体后的语句，立即进行下次的 WHILE 循环
CASE	根据条件列表返回多个可能结果之一的表达式
RETURN	无条件退出

IF 语句和 WHILE 语句中都需要指定条件，WHERE 子句中也需要指定条件，二者使用场合不同。IF 语句和 WHILE 语句在 T-SQL 程序中实现流程控制；而 WHERE 子句仅作为 SELECT 语句的一部分使用。

4.6 存储过程

存储过程是命名的 T-SQL 语句的集合，该对象在数据库中存储并作为独立单元被处理和在程序中使用。

SQL Server 数据库中有两种存储过程：系统存储过程和用户自定义存储过程。

4.6.1 系统存储过程

系统存储过程是一组 SQL Server 提供的存储过程，可用于执行一些操作，如从系统目录检索信息或执行管理任务。系统存储过程在 master 数据库中创建并存储，带有 sp_ 前缀。可从任何数据库中执行系统存储过程。下面是一些常见系统存储过程的介绍。

使用系统存储过程 sp_helpdb 显示服务器中的所有数据库及其属性。

```
EXEC sp_helpdb
```

使用系统存储过程 sp_helpdb 显示服务器中的“东山县医院财务库”数据库及其属性。

```
EXEC sp_helpdb 东山县医院财务库
```

使用系统存储过程 sp_help 显示数据库中的所有对象信息。

```
EXEC sp_help
```

使用系统存储过程 sp_help 显示东山县医院业务库中的医生表对象信息。

```
EXEC sp_help 医生
```

使用存储过程 sp_attach_db 将“东山县医院业务库”数据库加载到 SQL Server 中。

```
EXEC sp_attach_db @dbname='东山县医院业务库',@filename1='d:\data\东山县医院业务库.mdf'
```

使用系统存储过程 sp_renamedb 重命名数据库。

```
EXEC sp_renamedb '东山县医院', '东山县医院财务库'
```

4.6.2 用户自定义存储过程

实践中通常把完成特定功能的、经常使用的一段代码定义为存储过程使用。

要编写自己的存储过程，例如实践中经常用到的计算个人所得税的程序，那么就可以在查询编辑器中将该程序定义为存储过程，如图 4-10 所示。

创建存储过程的语句是 CREATE PROCEDURE。后面的 tax 是存储过程的名字。

第二行的“@S DECIMAL(7,2)”声明了存储过程的参数。

BEGIN 和 END 括起来的脚本为存储过程体，完成计算功能。

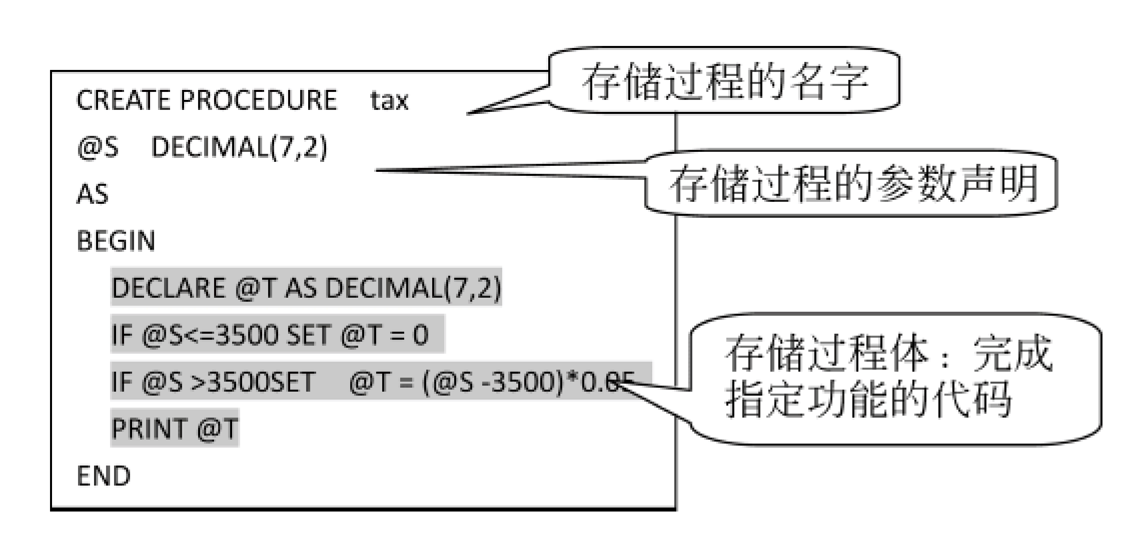


图 4-10　存储过程的组成

存储过程创建成功后，可以在 SQL 编辑器中通过 EXEC 调用，如：

```
EXEC tax 1800
```

tax 1800 的意思是以 1800 作为参数，调用存储过程 tax。“调用”的含义是首先把 1800 赋值给@S，然后让 SQL Server 转去执行存储过程体中的程序。

存储过程创建成功后，在“对象资源管理器”中右击“存储过程”组，选择“刷新”，就会看到该存储过程。

如果修改存储过程的定义，可以在管理器中右击存储过程名，在弹出的快捷菜单中选择“修改”选项，查看并修改过程体。

再如，设计存储过程，根据部门号显示该部门的门诊收费总和。

```
CREATE PROCEDURE BuMenShouFei @bmh varchar(10)
AS
SELECT  SUM(单价 * 数量) AS 部门收费总额
FROM 门诊收费明细
WHERE 门诊流水号 in (SELECT 门诊流水号 FROM 门诊收费 WHERE 部门号=@bmh)
```

上述存储过程创建成功后，在查询编辑器中可以调用该存储过程，计算 10201 部门的门诊收费总和。

```
EXEC  BuMenShouFei  '10201'
```

4.7　自定义函数

T-SQL 中提供了 left()等内置函数。这些函数可以在 SQL 中直接使用。除了使用系统提供的函数外，审计人员还可以根据需要自定义函数。函数是 SQL Server 中的数据库对象。用户自定义函数是存储在数据库中的一段 T-SQL 程序，可以被调用和返回值。用户自定义函数不能用于执行一系列改变数据库状态的操作，但它可以像系统函数一样在查询或存储过程等的程序段中使用。

例如，创建函数，用来简单计算个人所得税。

可按以下步骤在管理器中创建函数。

第一步，在对象资源管理器中展开“东山县医院业务库”，继续展开“可编程性”，然后在“函数”上右击，在弹出的快捷菜单中选择“新建”，然后选择“标量值函数”，如图 4-11 所示。

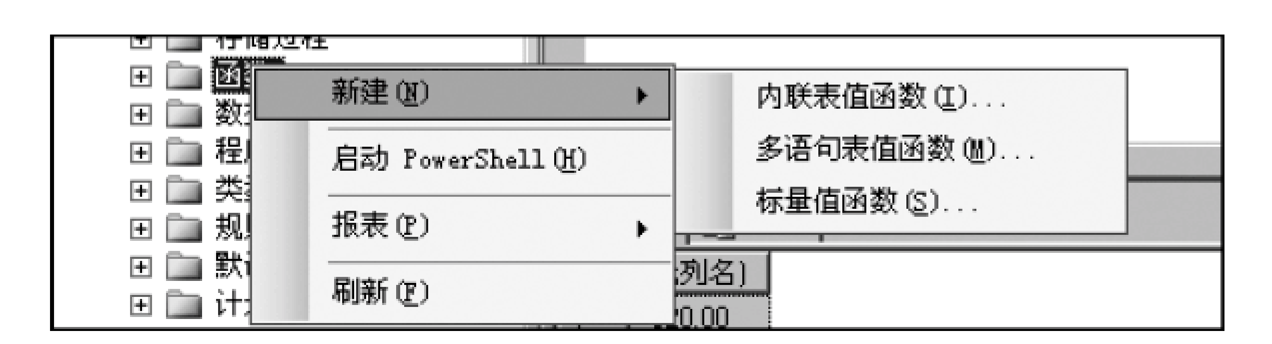


图 4-11　新建函数

第二步，在为“新建函数”打开的 SQL 编辑器中，如图 4-12 所示，把自动生成的程序模板删除，根据需要定义函数的名称、参数以及函数体如下。

```
CREATE FUNCTION  gsh (@S DECIMAL(7,2))
RETURNS DECIMAL(7,2)
AS
BEGIN
  DECLARE @T AS DECIMAL(7,2)
  IF @S<=3500 SET @T=0
  IF @S>3500 SET  @T=(@S-3500)*0.05
  RETURN @T
END
```

```
SQLQuery1.s...rator (56))
-- =============================================
-- Author:      <Author,,Name>
-- Create date: <Create Date, ,>
-- Description: <Description, ,>
-- =============================================
CREATE FUNCTION <Scalar_Function_Name, sysname, FunctionName>
(
    -- Add the parameters for the function here
    <@Param1, sysname, @p1> <Data_Type_For_Param1, , int>
)
RETURNS <Function_Data_Type, ,int>
AS
BEGIN
    -- Declare the return variable here
    DECLARE <@ResultVar, sysname, @Result> <Function_Data_Type, ,int>

    -- Add the T-SQL statements to compute the return value here
    SELECT <@ResultVar, sysname, @Result> = <@Param1, sysname, @p1>

    -- Return the result of the function
    RETURN <@ResultVar, sysname, @Result>

END
GO
```

图 4-12　函数体

第三步，写好后，单击“分析”按钮进行检查，如果无误，则单击“执行”按钮，函数创建成功。

函数创建成功后，在“对象资源管理器”中右击“函数”组，选择“刷新”，就会在“标量值函数”组中看到该函数。

在该函数定义中，各个组成部分的含义如图 4-13 所示。

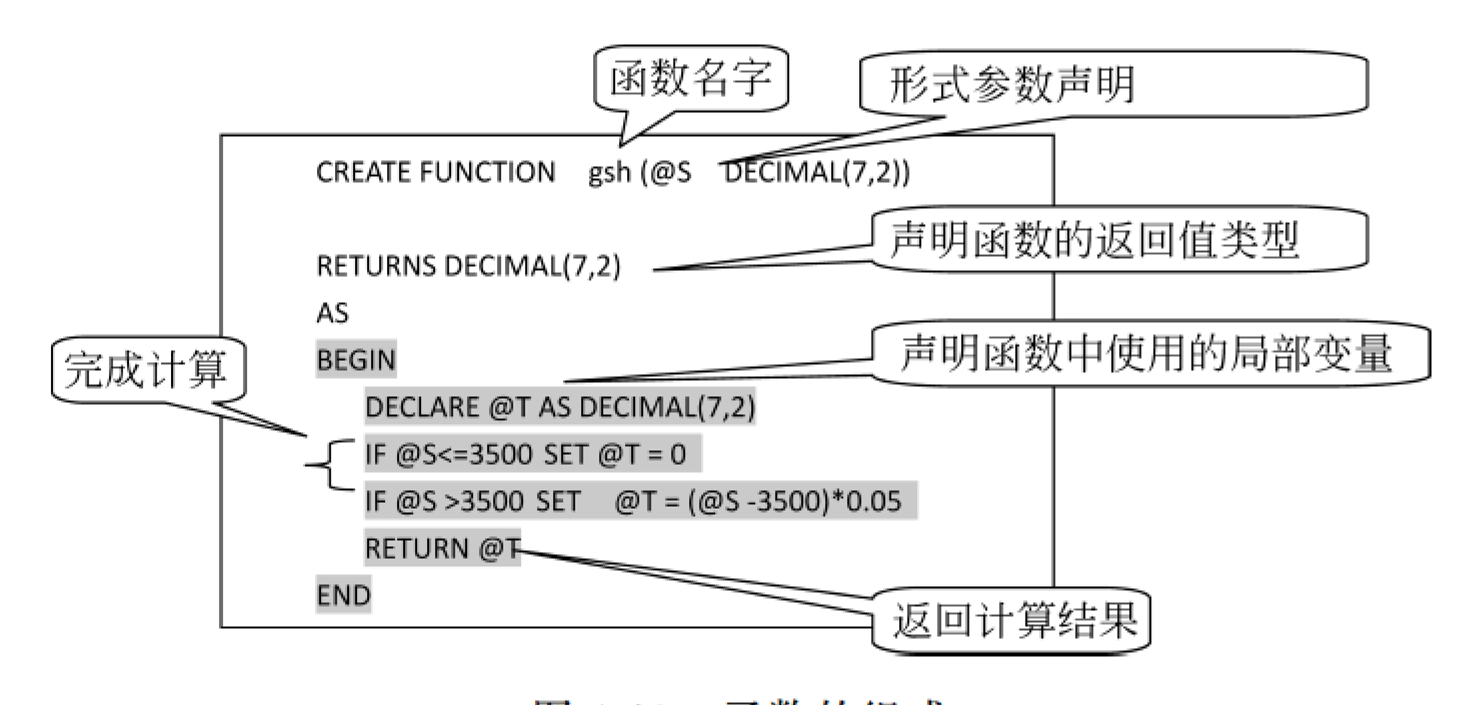


图 4-13　函数的组成

在当前数据库上新建查询，在打开的 SQL 编辑器中使用如下查询调用该函数。

```
SELECT dbo.gsh(1200)
```

这里，gsh(1200)称为函数调用。当发生函数调用时，首先把实在参数 1200 赋值给形式参数@S，然后让 SQL Server 转去执行完成计算的函数体（即图 4-13 中 BEGIN END 括起来的阴影部分）。函数体中的最后一条语句是 RETURN @T，该语句把变量@T 中存放的计算结果返回到函数调用位置，所以在 select gsh(1200) 语句执行时，首先计算 gsh(1200)得到结果 0，然后把 0 显示出来。这就好比是 select 1＋2 首先计算 1＋2，然后显示表达式 1＋2 的结果 3 一样。

若要修改和删除用户自定义函数，那么在管理器中选择要修改的用户自定义函数，右击，从弹出的快捷菜单中选择“修改”选项，则会出现“用户自定义函数”对话框。在其中可以修改用户自定义函数的函数体、参数等。从弹出的菜单中选择“删除”选项，则删除用户自定义函数。

若要查看用户定义的函数，则使用命令：

```
sp_help gsh
```

也可以直接使用 CREATE FUNCTION 创建函数。例如，定义计算 1—12 月中某月个人所得税的函数的语句如下。

```
/* 功能:计算 1—12 月的应交所得税函数
 * 输入:个人月收入
 * 返回值:月个人计税总额
 */
CREATE FUNCTION geShui(@income money)
RETURNS   money
AS
BEGIN
DECLARE @T   AS money
@T = CASE
     when @income<=3500 then 0
```

```
    when @income-3500>0 and @income-3500<=500 then (@income-3500)*0.05
    when @income-3500>500 and @income-3500<=2000 then
        (@income-3500)*0.1-25
    when @income-3500>2000 and @income-3500<=5000 then
        (@income-3500)*0.15-125
    when @income-3500>5000 and @income-3500<=20000 then
        (@income-3500)*0.2-375
    when @income-3500>20000 and @income-3500<=40000
then (@income-3500)*0.25-1375
    else null
    END
RETURN @T
END
```

假设收入为2500，则计算个税的语句为

```
SELECT geShui(2500)
```

假设在表 TEST 中有“月收入”列：

```
CREATE TABLE #TEST(月收入 money)
INSERT INTO #TEST VALUES(3000)
INSERT INTO# TEST VALUES(8000)
```

可在表上直接计算个税：

```
SELECT 月收入, geShui(月收入) AS 个人所得税
FROM #TEST
```

此例的另外一种脚本是直接使用 RETURN 返回计算结果：

```
CREATE FUNCTION geShui(@x money)
RETURNS  money
AS
BEGIN
RETURN CASE
    when @x<=3500 then 0
    when @x-3500>0 and @x-3500<=500 then (@x-3500)*0.05
    when @x-3500>500 and @x-3500<=2000 then (@x-3500)*0.1-25
    when @x-3500>2000 and @x-3500<=5000 then (@x-3500)*0.15-125
    when @x-3500>5000 and @x-3500<=20000 then (@x-3500)*0.2-375
    when @x-3500>20000 and @x-3500<=40000 then (@x-3500)*0.25-1375
    else null
    END
END
```

注意，在本例的函数体中只有一条语句（RETURN），该语句中包含了一个 CASE 表达式（从 CASE 开始，到 END 结束）。

可以在表达式中使用函数,而不能在表达式中使用存储过程。创建函数的语法为

```
CREATE FUNCTION  <函数名>([参数 1数据类型 1,参数 2数据类型 2,… ])
RETURNS <返回值数据类型>
AS
BEGIN
<函数体>
    RETURN <表达式>
END
```

一个函数最多可以定义 1024 个参数,每个参数前用“@”符号标明。参数的作用范围是整个函数。参数只能替代常量,不能替代表名、列名或其他数据库对象的名称。

参数的数据类型可以为除 TEXT、NTEXT、IMAGE、CURSOR、TIMESTAMP 和 TABLE 类型外的其他数据类型。

<函数体>是一系列的 T-SQL 语句,它们决定了函数的功能。

4.8 触 发 器

触发器是一种特殊类型的存储过程,它在指定的表中的数据发生变化时自动执行。唤醒调用触发器,以响应 INSERT、UPDATE 或 DELETE 语句。触发器可以查询其他表,并可以包含复杂的 T-SQL 语句。触发器是审计数据修改的重要设施。

触发器具有以下特点:它是在操作有效后才执行的,即其他约束,如 CHECK 约束,优先于触发器;它与存储过程的不同之处在于,存储过程可以由用户直接调用,而触发器不能被直接调用,是由事件触发的;一个表可以有多个触发器,在不同表上,同一种类型的触发器也可以有多个;触发器是定义在行更新语句上的,该语句可能影响多行,SQL Server 仅为每个更新语句触发,而不是为每一行触发。

SQL Server 提供了两种类型的触发器:INSTEAD OF 触发器和 AFTER 触发器。两种类型的触发器比较见表 4-2。

表 4-2 两种类型的触发器比较

触 发 器	INSTEAD OF	AFTER
DML 语句	模拟但不执行	执行,但可以在触发器中回退
执行时机	在 PK 和 FK 约束前	事务完成后提交前(所有约束满足后)
是否可应用于视图	是	否
每个表上可能的事件	一个	多个

当向表中插入数据时,所有数据约束都通过之后,INSERT 触发器就会执行。当对表删除数据时,DELETE 触发器会被执行。利用 UPDATE 修改一条记录时,相当于删除一条记录,然后再增加一条新记录。

例如,在表 T 上创建触发器,使得每插入一行后,显示“ONE ROW IS INSERTED.”。

首先创建表 T：

```
CREATE TABLE T ( A INT, B INT)
```

然后在 T 上创建触发器：

```
CREATE TRIGGER TRG_insert_rows
ON T
AFTERINSERT
AS
BEGIN
    PRINT 'ONE ROW IS INSERTED.'
END
```

现在测试该触发器。在查询编辑器中输入插入语句：

```
INSERT INTO T VALUES (11, 12)
```

会显示：

```
ONE ROW IS INSERTED.
(1 行受影响)
```

现在创建一个 INSTEAD OF 触发器：

```
CREATE TRIGGER TRG_INSTEADOF_insert_rows
ON T
INSTEAD OF INSERT
AS
BEGIN
    PRINT 'INSTEAD OF TRIGGER FOR INSERT.'
END
```

在对象资源管理器中右击 TRG_insert_rows 触发器，从弹出的快捷菜单中选择“禁用”选项，然后执行下面的插入语句。

```
INSERT INTO T VALUES (21, 22)
```

显示结果如下。

```
INSTEAD OF TRIGGER FOR INSERT.
(1 行受影响)
```

使用查询语句观察是否把(21,22)插入了：

```
SELECT *
FROM T
```

发现表中仍然只有一行，没有将(21,22)插入。

如果在会计科目表上定义了删除触发器 TRG_delete_rows：

```
CREATE TRIGGER TRG_delete_rows
ON 会计科目表
FOR DELETE
AS
BEGIN
    RAISERROR ('不允许删除会计科目表中的行',11,1)
            PRINT  'Hi'
END
```

然后执行删除语句,会发现出错消息:

```
服务器:消息 50000,级别 11,状态 1,过程 TRG_delete_rows,行 7
不允许删除会计科目表中的行
(所影响的行数为 0 行)
```

该错误信息中包含了5个部分。

错误号:每个错误消息都具有唯一的错误号;约定用户自定义消息的错误号大于50000。

严重性:严重性指示错误的严重程度。严重性较低的错误,如级别0～9,为信息性消息或低级警告;严重级别为11～16的错误信息由用户生成,并由用户修正。

状态:描述发生错误时的状态,对每个引发错误的特定情况都分配唯一的状态代码(1～127)。默认值是1。

过程名称:出现错误的存储过程或触发器的名称。

行号:指示在批处理、存储过程、触发器或函数中产生错误的语句所在行。

错误消息字符串:错误消息包含关于错误原因的诊断信息。

4.9 游　　标

由SELECT语句返回的行集包括所有满足该语句WHERE子句中条件的行。由语句返回的这一完整的行集被称为结果集。

应用程序,特别是交互式联机应用程序,并不总能将整个结果集作为一个单元有效地处理。这些应用程序需要一种机制,以便每次处理一行或一部分行。游标就是提供这种机制的结果集扩展。结果集在系统中的位置如图4-14所示。

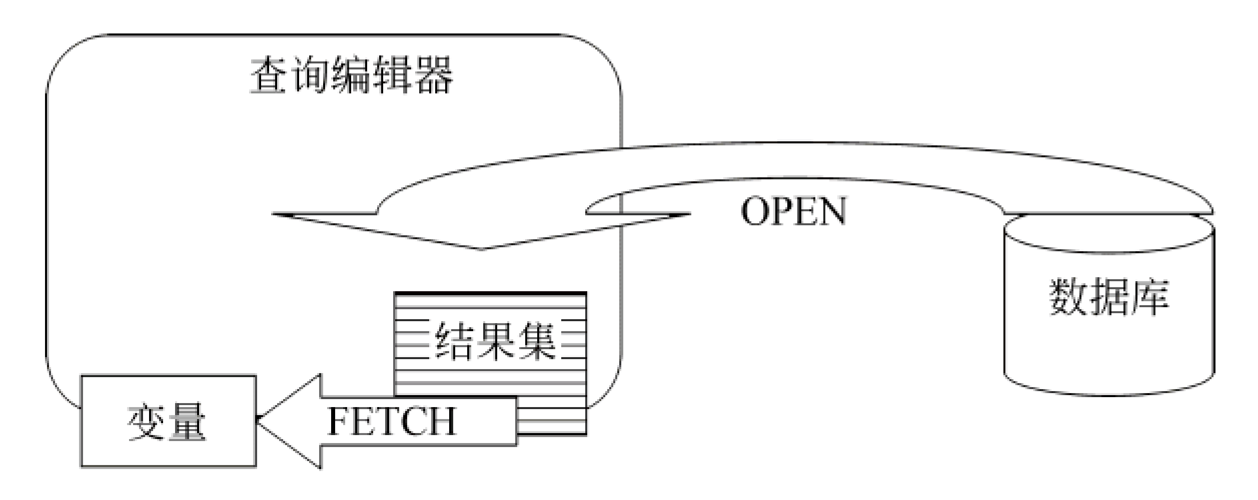


图4-14　结果集在系统中的位置

游标通过以下方式扩展结果处理：允许定位在结果集的特定行；从结果集的当前位置检索一行或多行；支持对结果集中当前位置的行进行数据修改。

使用游标的一般过程：①创建游标；②打开游标；③重复从游标中提取(FETCH)行进行处理，直到到达末尾；④关闭游标；⑤释放游标。

例如，在程序中逐行提取医生表中的各行，依次存入程序变量中。

(1) 创建游标。

```
DECLARE cur_医生  CURSOR
FOR SELECT  *  FROM 医生
```

现在有了名为“cur_医生”的游标，游标中包含了医生表中的所有记录，但在使用这些记录前，须先打开游标。

(2) 打开游标。

```
OPEN cur_医生
--"cur_医生"是游标名称
```

(3) 滚动游标，提取数据。

下面的语句从游标中提取数据，并放到程序变量中。

```
DECLARE @医生号 char(30)
DECLARE @姓名 char(40)
DECLARE @部门号 char(20)
FETCH cur_医生 INTO  @医生号,@姓名,@部门号
```

可以使用 WHILE 循环从游标中提取所有记录，但如何判断到达了游标的末尾？全局变量@@FETCH_STATUS 记载最近一次提取语句(FETCH)的执行状态，其取值及含义如下。

返回值	描述
0	FETCH 语句成功
-1	FETCH 语句失败或此行不在结果集中
-2	被提取的行不存在

全局变量@@ROWCOUNT 返回受上一语句影响的行数，包括 FETCH。

```
DECLARE @医生号 char(30)
DECLARE @姓名 char(40)
DECLARE @部门号 char(20)
FETCH cur_医生 INTO  @医生号,@姓名,@部门号
WHILE (@@FETCH_STATUS = 0)
BEGIN
  print @医生号
  print @姓名
  print @部门号
  FETCH cur_医生 INTO  @医生号,@姓名,@部门号
END
```

（4）关闭游标。

```
CLOSE  cur_医生
```

游标被关闭后依然存在，以便再次被打开。当不再使用游标时，应当释放它。

（5）释放游标。

```
DEALLOCATE cur_医生
```

与表、视图、索引、存储过程和触发器不同，游标并不是数据库对象。游标可在3个范围内有效。游标在连接中有效。连接开始于用户登录（log on），结束于用户注销（log off）。当用户登录后创建了游标，则该游标在用户注销后失效。在存储过程执行期间，存储过程中定义的游标有效。触发器中创建的游标仅在触发器执行期间有效。

下面的例子演示如何应用游标实现随机抽样。

为了减少数据规模（一般限制在50000行记录），降低审计风险，通常使用的技术之一是随机抽样。

假设对"银行贷款"数据库中的"借款凭证表"进行10％随机抽样，原表中有114460行。首先在被抽样表中增加行号和随机数两列，然后为每一列指派一个随机数，接着按照随机数列对所有行排序，最后从排序结果中提取前10％。脚本如下。

```
/* 使用临时表,以提高性能 */
  SELECT *
  INTO##被抽样表
  FROM借款凭证表
/* 修改临时表的表结构,以向其中插入随机数 */
  ALTER TABLE  ##被抽样表
  ADD  行号 int, 随机数 int
/* 使用游标为每一行指派随机数 */
  DECLARE @row int;
  DECLARE U_C CURSOR
  FOR SELECT *  FROM  ##被抽样表
  FOR UPDATE OF 行号,随机数;
  OPEN U_C;
  FETCH NEXT FROM U_C;
  SET @row=0;
  WHILE @@FETCH_STATUS=0
  BEGIN
    UPDATE ##被抽样表
    SET 行号 =@row, 随机数 =cast(floor(rand() * 100000000) AS int)
    WHERE CURRENT OF U_C;
    SET @row=@row+1;
    FETCH NEXT FROM U_C;
  END
  CLOSE U_C
  DEALLOCATE U_C
```

```
/* 从添加了随机数的临时表中选取 10% 作为抽样结果 */
  SELECT top 10 percent  *
  INTO 抽样表
  FROM# #被抽样表
  ORDER BY 随机数 ASC
```

也可以使用以下简单的查询实现随机抽样。

```
SELECT top 10 percent *
FROM 借款凭证表
ORDER BY newid()
```

如果 Microsoft SQL Server 是后面的更高的版本，以下语句的抽样执行效率最高。

```
SELECT *
FROM 借款凭证表
TABLESAMPLE (10 PERCENT)
```

注意：TABLESAMPLE 是在 SQL Server 2005 中引入的。当 TABLESAMPLE 用于从早期版本升级的数据库时，数据库的兼容级别必须至少设置为 90。设置数据库兼容级别如图 4-15 所示。

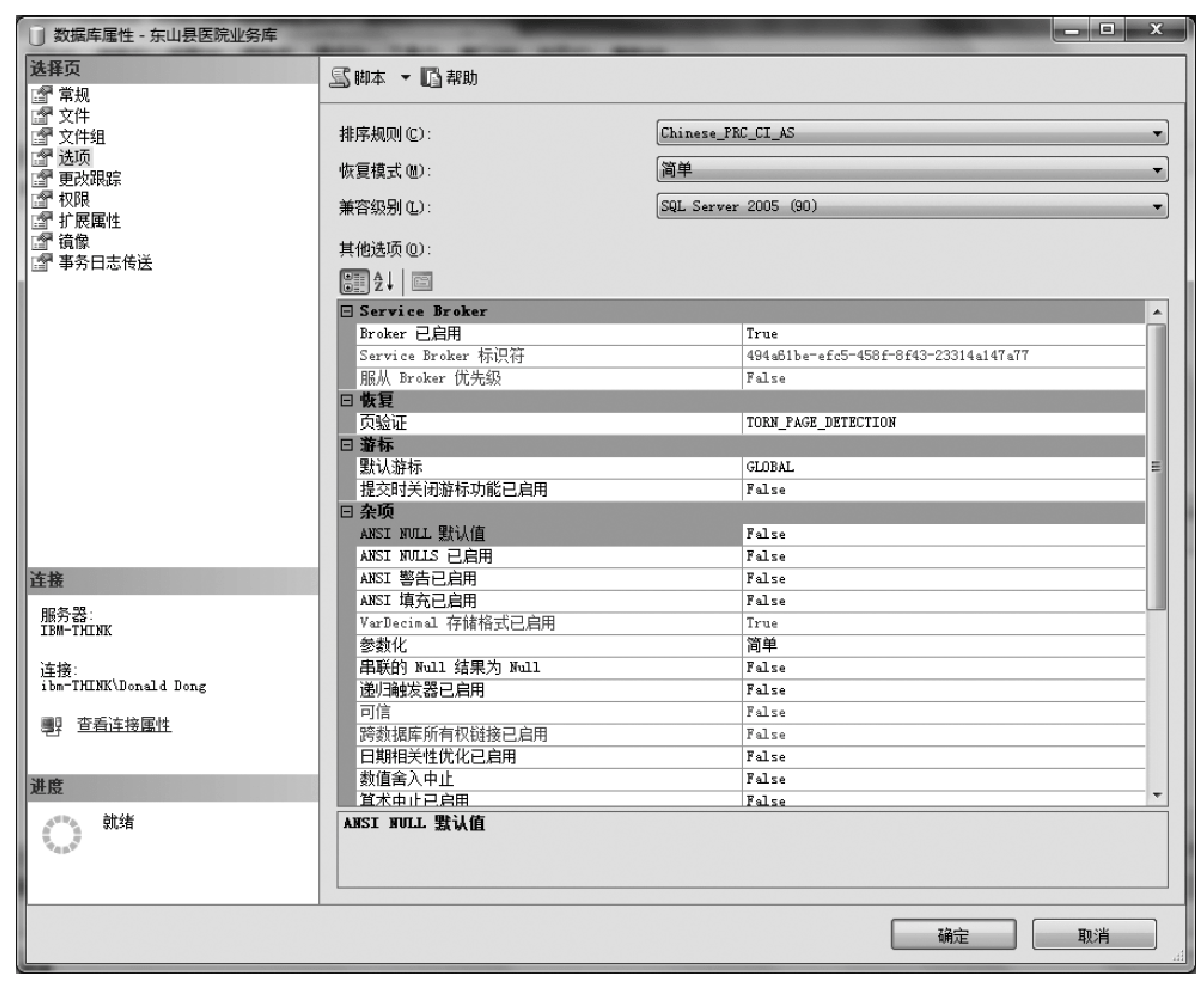


图 4-15　设置数据库兼容级别

使用游标进行逐个记录更新时，须选择 SQL 编辑器以文本方式显示结果，否则系统会出现“资源不足”问题。

4.10 事务与并发控制

在多用户和网络环境下，数据库是一个共享资源，多个用户或应用程序同时对数据库的同一数据对象进行读写操作，这种现象称为对数据库的并发操作。显然，并发操作可以充分利用系统资源，提高系统效率。虽然如此，但是如果对并发操作不进行控制，就会造成一些错误。对并发操作进行的控制称为并发控制。并发控制能力是衡量一个 DBMS 的重要性能指标之一。

4.10.1 事务的概念

事务(transaction)是用户定义的一个数据库操作序列，这些操作要么全做，要么全不做，是一个不可分割的工作单位。例如，银行转账的业务过程是：从 1001 账户划走 800 元到 1002 账户。这笔业务对应两个数据库操作：1001 账户减去 800 元和 1002 账户增加 800 元。这两个数据库操作不可分割。

在 SQL Server 中，事务的开始与结束可以由用户显式定义。如果用户没有显式地定义事务，则由 DBMS 按默认自动划分事务。在 SQL 中，定义事务的语句有 3 条：BEGIN TRANSACTION、COMMIT TRANSACTION、ROLLBACK TRANSACTION。

事务通常以 BEGIN TRANSACTION 开始，以 COMMIT TRANSACTION 或 ROLLBACK TRANSACTION 结束。COMMIT 的作用是提交，即提交事务的所有操作。事务提交是将事务中所有对数据的更新写回到磁盘上的物理数据库中，事务正常结束。ROLLBACK 的作用是回滚，即在事务运行的过程中发生了某种故障，事务不能继续执行，系统将事务中对数据库的所有已完成的操作全部撤销，回滚到事务开始时的状态。

事务具有 4 个特性，即原子性、隔离性、一致性和持续性。

原子性(atomicity)：事务中包括的诸操作要么都做，要么都不做。也就是说，事务作为一个整体单位被处理，不可以被分割。

隔离性(isolation)：一个事务的执行不能被其他事务干扰，即一个事务内部的操作及使用的数据对其他并发事务是隔离的，并发执行的各个事务之间不能互相干扰。

一致性(consistency)：事务执行的结果必须使数据库处于一个一致性状态。当数据库中只包含成功事务提交的结果时，就说数据库处于一致性状态。如果数据库系统运行中发生故障，有些事务尚未完成就被迫中断，这些未完成事务对数据库所做的修改有一部分已写入物理数据库，这时数据库就处于一种不正确的状态，或者说是不一致状态，数据库系统必须确保事务的一致性。

持续性(durability)：也称永久性(permanence)。持续性指一个事务一旦提交，它对数据库中数据的改变就是永久性的，接下来的其他操作或故障不应该对其执行结果有任何影响。

事务的这些特性由数据库管理系统中的并发控制机制和恢复机制保障。

4.10.2 事务类型

SQL Server 的每一个连接可以使用它们需要的事务模式实现它们的需求:自动提交事务、显式事务和隐式事务。

由 SQL Server 定义的事务称作自动提交事务。SQL Server 将一切都作为事务处理。用户没有定义事务,SQL Server 会自己定义事务。例如:

假设存放账户的表 account 的结构如下。

```
列名    类型
账户    char(4)
金额    int
```

创建该表的 SQL 语句如下。

```
CREAT table account(账户 char(4),金额 int CHECK (金额>=0))
```

表中的数据如下。

账　户	金额/元	账　户	金额/元
1001	700	1002	2600

而且在金额列上具有 CHECK 约束:金额>=0。

那么,完成从 1001 账户到 1002 账户转账 800 元事务的 SQL 语句是:

```
--操作 1:从 1001 账户上减 800 元
  UPDATE account
  SET  金额 =金额 - 800
  WHERE 账户 ='1001'
--操作 2:从 1002 账户上加 800 元
  UPDATE account
  SET  金额 =金额 + 800
  WHERE 账户 ='1002'
--操作 3:查看结果
  SELECT *
  FROM account
```

在 SQL 查询分析器中打开"查询"菜单,选择"文本显示结果"。在显示结果窗格会看到第一个操作没有成功,因为 700-800 后账户金额小于 0,违反了该表上的 CHECK 约束。但是,第二个操作和第三个操作被成功完成。**SQL Server 使用自动提交事务时,每一个语句本身是一个事务**。如果这个语句产生了错误,它的事务会自动回滚。如果这个语句成功执行而没有产生错误,它的事务会自动提交。因此,第二个和第三个操作将被提交,而第一个有错误的操作会回滚。

在显式事务中,用户要定义一个事务在何处开始,并定义这个事务在什么时候需要提交或回滚。这通过 T-SQL 语句 BEGIN TRANSACTION、COMMIT TRANSACTION

和 ROLLBACK TRANSACTION 实现。

现在希望将 3 个操作要么都完成,要么一个也不做。因此需要将各操作组织在一个显式事务中,而且指明若其中任意一个操作发生意外,就回退全部操作(整个事务)。在查询窗口中输入以下语句并一次执行所有语句。

```
--下面定义错误出现后回退事务,约束 3条语句要么全做,要么全不做
  DECLARE @BACK int
  SET @BACK = 0
  BEGIN TRAN
--操作 1:从 1001 账户上减 800 元
  UPDATE account
  SET  金额 =金额 - 800
  WHERE 账户 = '1001'
  IF @@ERROR < > 0
  SET @BACK = 1
--操作 2:从 1002 账户上加 800 元
  UPDATE account
  SET  金额 =金额 + 800
  WHERE 账户 = '1002'
  IF @@ERROR < > 0
    SET @BACK = 1
--操作 3:查看结果
  SELECT *
  FROM account
  IF @@ERROR < > 0
    SET @BACK = 1
  IF @BACK = 1
  BEGIN
--若 3条语句其中之一失败,就回退事务
  PRINT '回退事务'
  ROLLBACK TRAN
    END
    ELSE
    BEGIN
--3条语句均成功,提交事务
    COMMIT TRAN
    END
```

当 SQL Server 完成 T-SQL 语句的执行时,如果语句执行成功,则@@ERROR 设置为 0。若出现一个错误,则返回一条错误信息。@@ERROR 返回此错误信息代码,直到另一条 T-SQL 语句被执行。可以在 sysmessages 系统表中查看与@@ERROR 错误代码相关的文本信息。

由于@@ERROR 在每一条语句执行后被清除并且重置,所以应在语句验证后立即检查它,或将其保存到一个局部变量中,以备事后查看。

```
--检查结果
SELECT *
FROM account
```

可以看出，整个事务都回滚了。

上例中通过在创建表时，对金额做了“>=0”的 CHECK 限制，这样就可以通过全局变量@@ERROR 判断 SQL 语句是否出错，进而判断事务是否应该提交或者回滚。如果上例中金额一列并没有该 CHECK 限制，那就不能通过@@ERROR 变量判断了。在这种情况下，可以直接在 SQL 语句中自己判断账户余额是否小于 0。

```
BEGIN TRAN
--操作 1:从 1001 账户上减 800 元
UPDATE account
SET 金额 =金额-800
WHERE 账户 ='1001'
IF(SELECT 金额 FROM account WHERE 账户 ='1001')<0
BEGIN
  PRINT '回退事务'
  ROLLBACK TRAN
END
ELSE
BEGIN
--操作 2:从 1002 账户上加 800 元
UPDATE account
SET 金额 =金额 +800
WHERE 账户 ='1002'
COMMIT TRAN
END
--操作 3:查看结果
SELECT *
FROM account
```

第 3 种模式称作隐式事务模式。因为在这种模式中，SQL Server 在没有事务存在的情况下会开始一个事务，但不会像在自动模式中那样自动执行 COMMIT 或 ROLLBACK 语句。事务必须显式结束。以下语句隐式开始一个事务：ALTER TABLE、GRANT、FETCH、DELETE、CREATE、REVOKE、INSERT、SELECT、DROP、OPEN、UPDATE、TRUNCATE TABLE。

设置连接为隐式事务模式的语句为

```
SET IMPLICIT_TRANSACTIONS ON
```

可用@@TRANCOUNT 测试是否已经打开一个事务。测试方案如下：在创建表前先查询活动事务数，然后创建表；再查询活动事务数；然后做一条插入语句；再查询活动事务数；最后关闭隐式事务。

```
--查询活动事务数,此时为 0。该语句测试当前连接的活动事务数
--结果是 1 的意思是当前连接已经打开了一个事务;结果是 0 的意思是当前没有事务
  SELECT @@TRANCOUNT
--创建表,开始一个隐式事务
  CREATE TABLE T1 (ID int PRIMARY KEY)
--再次测试当前连接的活动事务数,结果是 1
  SELECT @@TRANCOUNT
  INSERT INTO T1 VALUES(5)
--再次测试当前连接的活动事务数,结果是 1,由于已经有一个打开的事务,因此 SQL Server 没有开始一个新的事务
  SELECT @@TRANCOUNT AS [Transaction Count]
--执行以下语句回滚这个事务,并再次检查@@TRANCOUNT。可以看出,在 ROLLBACK TRAN 语句执行之后,@@TRANCOUNT 的值变成了 0
  ROLLBACK TRAN
  SELECT @@TRANCOUNT
--执行以下代码关闭隐式事务
  SET IMPLICIT_TRANSACTIONS OFF
```

由于没有显式的 BEGIN TRANSACTION 语句,提交或回滚步骤很容易被忘记,导致事务长期运行,或在连接关闭时产生不必要的回滚,以及与其他连接之间产生阻塞问题。

4.10.3 并发操作可能产生的问题

假设存放账户的表 account 的结构如下。

列名	类型
账户	char(4)
金额	int

表中的数据如下。

账　　户	金额/元
1001	1800
1002	2600
…	

这里以账户管理为例,说明对并发操作不加以限制,会产生数据不一致性问题,这种问题共有 4 类:丢失更新、脏读、不可重复读、幻象读。

当两个或多个事务选择同一行,然后基于最初选定的值更新该行时,会发生丢失更新问题。每个事务都不知道其他事务的存在。最后的更新将覆盖由其他事务所做的更新,这将导致数据丢失。

假设账户 1001 的金额为 1800 元,现在甲执行一个事务存入 200 元,乙执行另一个事务取出 400 元。

同时发生存入(甲)和取出(乙)操作，这就形成并发操作。一种可能的并发操作次序见表 4-3，发生了“丢失更新”错误。

表 4-3　发生丢失更新的过程

顺　　序	事　　务	操　　作	金额/元
1	甲	读	1800
2	乙	读	1800
3	甲	AMOUNT＝1800＋200	
4	乙	AMOUNT＝1800－400	
5	甲	写	2000
6	乙	写	1400

当甲和乙并发执行时，在甲对数据库更新的结果没有提交之前，乙使用了甲的结果，而在乙操作之后甲又回滚，这时引起的错误是乙读取了甲的“脏数据”。表 4-4 所示的执行过程就产生了这种错误。

表 4-4　“脏读”过程

顺　　序	事　　务	操　　作	金额/元
1	甲	读	1800
2	甲	AMOUNT＝1800＋200	
3	甲	写	2000
4	乙	读	2000
5	乙	AMOUNT＝2000－400	
6	甲	ROLLBACK	1800
7	乙	写	1600

当事务甲多次访问同一行而且每次读取同一数据时，可能会发生不一致的问题，因为事务乙也可能正在更改甲事务正在读取的数据。因而该行被认为不可重复读取。“不可重复读”的过程见表 4-5。

表 4-5　“不可重复读”的过程

顺　　序	事　　务	操　　作	金额/元
1	甲	读 AMOUNT＝1800	1800
2	乙	读 AMOUNT＝1800	
3	乙	写 AMOUNT＝AMOUNT－400	1400
4	甲	读 AMOUNT＝1400(同一事务，两个读取结果)	

当对某行执行插入或删除操作，而该行属于某个事务正在读取的行的范围时，会发生

幻象读问题。事务第一次读的结果集其中一行已不复存在于第二次读或后续读中,因为该行已被其他事务删除。同样,由于其他事务的插入操作,事务的第二次或后续读结果集有一行已不存在于原始读中。

并发操作之所以产生错误,是因为任务执行期间相互干扰造成的。当将任务定义成事务,事务具有的特性(特别是隔离性)得以保证时,就会避免上述错误发生。但是,如果只允许事务串行操作,会降低系统的效率。所以,多数 DBMS 进行并发控制,既保证了数据的一致性,又保障了系统效率。

4.10.4 隔离级别

为了遵守 ACID 规则,一个事务必须与其他事务相隔离。这意味着在一个事务中使用的数据必须与其他事务相隔离。为了实现这种分离,每一个事务都会锁住它使用的数据,以防止其他事务使用它。锁定义在需要锁定的资源上,这些资源可以是索引、数据行或者表。SQL Server 总会尝试小粒度地锁住资源。在大多数情况下,它会首先基于行级加锁。如果锁住的行太多,会自动提升锁至表级,这个过程是自动完成的。

SQL Server 通过使用不同类型的锁隔离事务。定义事务内容以及应在何种情况下回滚至关重要,定义如何以及在多长时间内在事务中保持锁定也同等重要,这由隔离级别决定。事务准备接受不一致数据的级别称为隔离级别。隔离级别是一个事务必须与其他事务进行隔离的程度。较低的隔离级别可以增加并发,但代价是降低数据的正确性。相反,较高的隔离级别可以确保数据的正确性,但可能对并发产生负面影响。应用不同的隔离级别,为每一个单独事务定义与其他事务的隔离程度。事务隔离级别的定义如下。

未提交读(READ UNCOMMITTED,事务隔离的最低级别)。在读数据时不会检查或使用任何锁,因此,在这种隔离级别中可能读取到没有提交的数据。

提交读(READ COMMITTED,SQL Server 默认级别)。只能读取已经提交的数据。某事务读数据期间,其他事务可以读取相同的数据,但不能更新;在某事务更新数据期间,其他事务不能读取同样的数据。

可重复读(REPEATABLE READ)。同已提交读级别那样读数据,但某事务读完数据后并不立即允许其他事务更新相同的数据,而是整个事务结束后才允许其他事务更新同样的数据。

可串行读(SERIALIZABLE,事务隔离的最高级别,事务之间完全隔离)。其工作方式类似于可重复读。但它不仅会锁定读取的数据行,还可能会锁定整个表,这样就阻止了新数据的插入,从而可以防止幻象读。隔离级别与数据不一致问题见表 4-6。

表 4-6 隔离级别与数据不一致问题

隔离级别	脏 读	丢失更新	不可重复读	幻象读
未提交读	是	是	是	是
提交读	否	是	是	是

续表

隔离级别	脏　　读	丢失更新	不可重复读	幻象读
可重复读	否	否	否	是
可串行读	否	否	否	否

无论定义什么隔离级别，某事务对数据的更改总是等到事务结束后才允许其他事务对这些数据进行读取和更改。打开管理器的"工具"菜单，选择"选项"，在"查询执行/SQL Server / 高级"中可以设置事务的隔离级别，如图 4-16 所示。默认的隔离级别是 READ COMMITTED（提交读）。

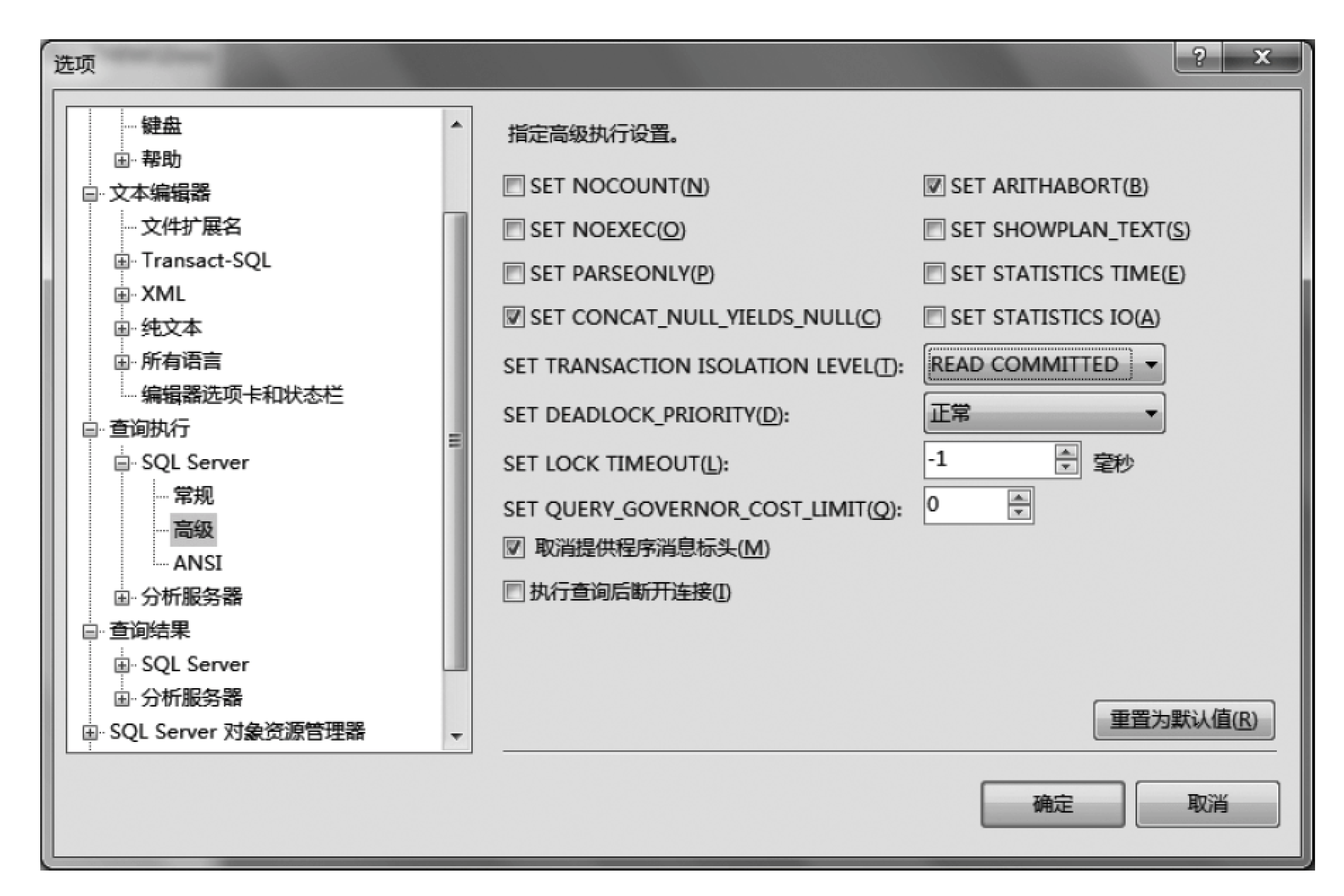


图 4-16　设置隔离级别

下面的操作步骤演示了"提交读"隔离级别下的并发行为。假设用户甲输入并执行以下语句，在 account 表中读取账户 1001 的金额。

```
BEGIN TRAN
SELECT 账户，金额
FROM  account
WHERE 账户 ='1001'
```

现在用户乙启动另一事务在用户甲事务打开状态下更改了金额。打开第二个查询窗口并执行以下语句，但不提交事务。

```
BEGIN TRAN
UPDATE account
SET 金额 =金额 +200
WHERE 账户 ='1001'
```

这个 UPDATE 语句会正常运行，因为已提交读级别中甲事务对数据读取 1001 账户完毕，就立即允许乙更新 1001 账户了，而并不等待整个事务结束。

用户甲再次查询：

```
SELECT 账户，金额
FROM  account
WHERE 账户 ='1001'
```

在提交读级别(默认级别)，由于乙正在对账户 1001 进行更新，事务尚未提交，所以甲的 SELECT 语句被阻塞。这个阻塞直到乙的事务结束。

用户乙执行一个 ROLLBACK TRAN 语句回滚 UPDATE 语句。

```
ROLLBACK TRAN
```

在用户乙结束事务后，用户甲的查询完成了，并且其结果与以前的一样。用户乙的事务结束，用户甲的查询不再被阻塞。由于用户乙的事务回滚，因此用户甲得到的结果是原来的数据。如果用户乙的事务被提交，则用户甲会得到新的数据并将其作为结果。

可以看出，在已提交读级别(SQL Server 默认)中，SQL Server 会等到含有更新操作的事务结束之后再允许其他事务进行读操作，以此获取真正的提交数据。还可以看出，某事务读取数据操作结束之后就立即允许其他事务对这些数据进行更新操作，而不持续到含有读操作的事务提交之后。由于数据更新事务造成阻塞，因此读数据会非常慢。

“提交读”级别的一个缺点是，一个事务读取的数据在事务运行期间可能被另一个事务更改。因此，在“提交读”级别下，不能保证在一个事务中读取的数据始终一样。

再次执行上面的过程，不同的地方是，让用户乙提交事务，然后让用户甲再次读取账户 1001，再提交自己的事务，发现甲在一个事务中对账户 1001 读取的结果前后不一致。

下面设置使用“可重复读”隔离级别，防止含有读操作的事务结束前其他事务更改正在读的数据。

用户甲在查询窗口中执行以下代码：

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
SELECT 账户，金额
FROM account
WHERE 账户 ='1001'
```

用户乙在另一个查询窗口执行以下代码进行更新：

```
BEGIN TRAN
UPDATE account
SET account =account +200
WHERE 账户 ='1001'
COMMIT
```

因为处于可重复读隔离级别，含有读操作的事务会防止其他事务更改数据，所以用户乙的更新会被阻塞。

现在用户甲再次执行如下查询：

```
SELECT 账户，金额
FROM account
WHERE 账户 ='1001'
```

因为用户乙的更新被阻塞，所以用户甲两次查询的结果相同。

现在用户甲提交，发现用户乙也提交了。

在“可重复读”隔离级别情况下，用户甲在查询窗口中执行以下代码：

```
BEGIN TRAN
SELECT 账户，金额
FROM account
WHERE 账户 ='1001'
```

用户乙打开第二个查询窗口并执行以下代码尝试插入：

```
INSERT INTO account VALUES('1001', 800)
```

即使正处于“可重复读”隔离级别，这个语句也会成功执行。因为可重复读只禁止在本事务提交前其他事务更新本事务内读操作涉及的数据。但 INSERT 语句向数据库中插入新数据，这是允许的。新行处于用户甲的事务中 SELECT 语句的查询范围之内，所以会在事务下一次获取相同条件数据的时候被读取到。这种情况称作“幻象读”。

用户甲重复 SELECT 语句并提交这个事务，代码如下所示。

```
SELECT 账户，金额
FROM account
WHERE 账户 ='1001'
COMMIT TRAN
```

可以观察到，新行被 SELECT 语句读取了，因为它处于这个语句的查询范围之内。这就是造成用户甲的同一事务中两次查询相同却得到不同的结果集(幻象读)。为了防止这种现象，可以使用“可序列化”隔离级别。这种隔离级别比可重复读级别更严格，不仅会锁定事务读取的数据，还会锁定事务的读取范围。但是，不是万不得已，一般不使用可序列化隔离级别。

假设希望用户甲和以前一样查看同样的金额，现在使用“可序列化”隔离级别阻止更新数据，并且阻止“幻象读”。执行以下查询：

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
BEGIN TRAN
SELECT 账户，金额
FROM account
WHERE 账户 ='1001'
```

用户乙打开第二个查询窗口插入一个新行：

```
INSERT INTO account VALUES('1001',800)
```

发现用户乙的插入被阻塞。

使用下面语句回到默认的"提交读"隔离级别。

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
```

4.11 在审计脚本语言中应用 SQL 语句

在审计过程中，审计人员对某类问题，如检查部门项目支出预算草案编制程序的合规性，检查部门项目支出预算草案编制程序的真实性，检查部门项目支出预算草案编制程序的合理性，检查部门项目支出预算草案编制程序的完整性等，形成了较为成熟的审计数据查询分析方法，这些方法由不止一条 SQL 语句组成，往往涉及中间结果和逐条分析等。而且这些问题均具有相同或者类似的业务模型。那么，这些审计方法就可以使用 ASL（审计脚本语言）表述出来，存储在审计师办公软件（Auditor Office，AO）系统中。当遇到同类型的审计项目或者类似的审计需求时，审计人员就可以从系统中保存的审计方法中选择一个作为参考，或者在自己的电子数据上直接运行该方法。

一个 ASL 程序通常由说明部分和语句部分组成。

1. 说明部分

程序中用的常量、变量，或类型及过程与自定义函数，需在使用之前预先说明。程序中的 VAR CurQuery，EndofResult，CurDiff；是变量说明。在当前计算机上，被处理的数据存放在变量中，实现处理的语句（指令）完成对变量中数据的计算或者变换，处理的结果往往需要保存在磁盘这样的外部设备上。好比是使用算盘：数据安排在算盘上，算盘的操纵者根据预设的方法（规则、步骤）不断变换算盘上的数据，计算的结果写到账本（外部设备）上。

变量说明的作用就是在计算机中给数据安排存放的地盘。VAR CurQuery，EndofResult，CurDiff；安排了 3 个地盘，分别以 CurQuery，EndofResult 和 CurDiff 标识。

除了说明变量外，还可以使用 CONST 声明常量，如 CONST PI＝3.1415926；使用 ARRAY 声明数组。

2. 语句部分

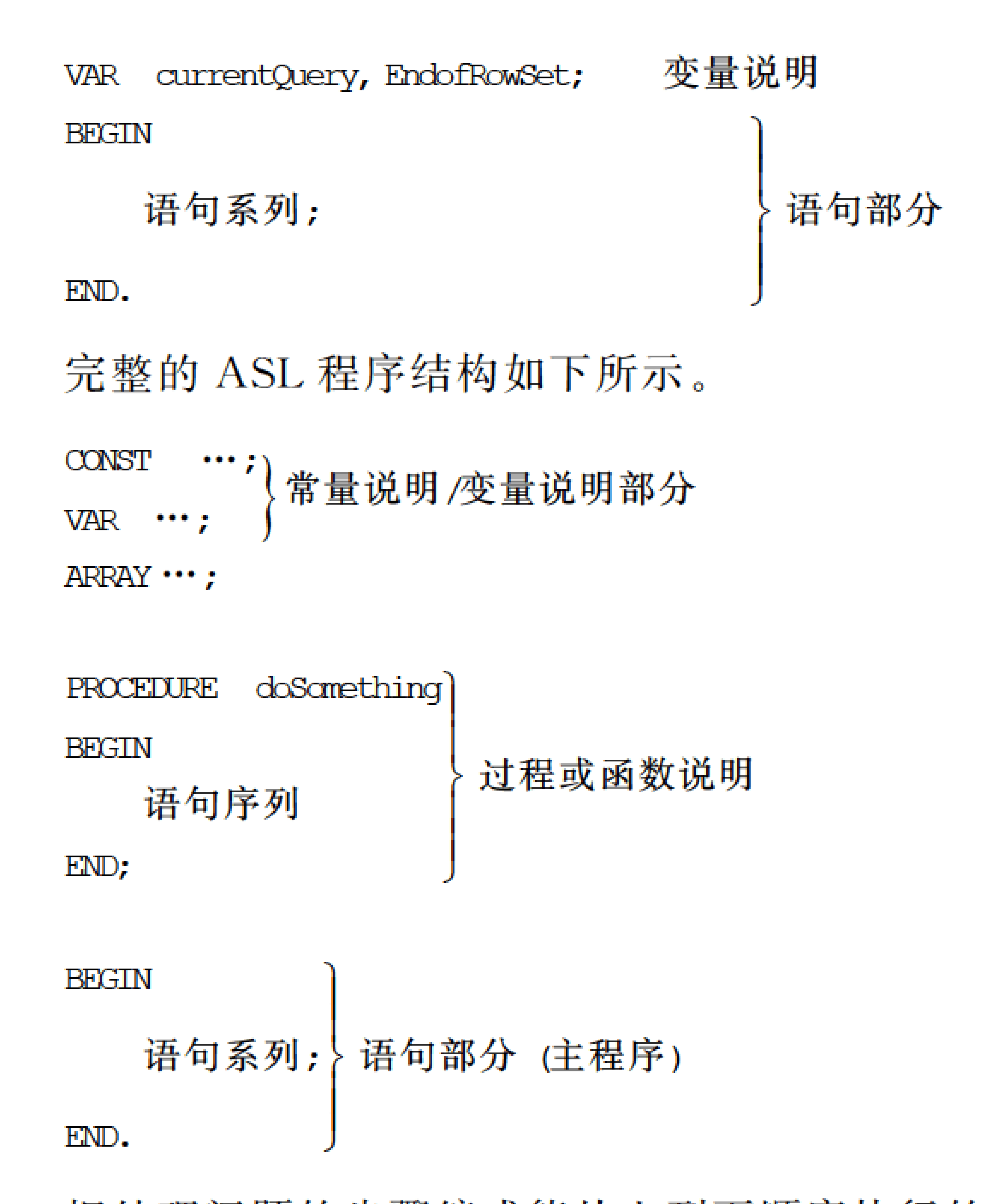
语句部分指保留字 BEGIN（开始）至 END.（结尾）之间的语句系列，这些语句是解决问题的具体处理步骤，将被计算机从上至下地顺序执行。

ASL 脚本程序不管是哪部分，每句末尾都必须有分号（；），程序结束的 END 末尾必须有圆点（.），是整个程序的结束标志。

程序中"//"到行末之间的部分为程序注释。注释是给审计人员看的，计算机完全不理会这些内容。注释非常重要，并非可有可无。注释以人的语言表达了审计方法的思路，易于理解。注释为其他审计人员阅读或者日后自己阅读程序均提供了基础，否则很容易

发生若干年后自己看不懂自己的程序的情况。

典型的 ASL 程序结构如下所示。



```
VAR  currentQuery, EndofRowSet;      变量说明
BEGIN
    语句系列;                          } 语句部分
END.
```

完整的 ASL 程序结构如下所示。

```
CONST  …;  } 常量说明/变量说明部分
VAR  …;
ARRAY …;

PROCEDURE  doSomething
BEGIN
    语句序列                 } 过程或函数说明
END;

BEGIN
    语句系列;  } 语句部分 (主程序)
END.
```

把处理问题的步骤编成能从上到下顺序执行的程序，是程序的基本特征。

AO 的【审计分析】|【审计方法】|【审计方法管理】为审计人员添加新的审计方法、参考和使用已有的审计方法提供了环境。审计人员可使用 ASL 编写审计方法作为"审计师"方法保存。

AO 系统中的每个审计方法都包括一个或者若干个审计步骤，每个审计步骤就是一段 ASL 程序；每个审计步骤中可以包括一个或者若干个 SQL 语句，这些 SQL 语句就是每段 ASL 程序的核心。

4.11.1 ASL 中的运算符

审计脚本语言中的运算符有赋值运算符、算术运算符、关系运算符和逻辑运算符等。

1. 赋值运算符

赋值运算符使用"：＝"，表示把该运算符右侧表达式的值存入左侧的变量中。例如：

```
InputNumber := 123;          //数值赋值,读作:把 123 赋值给变量 InputNumber
InputString := 'ABCDEFG';    //字符串赋值
```

注意，变量必须先声明，后使用。

2 算术运算符

ASL 中的算术运算符见表 4-7。

表 4-7 算术运算符

算术运算符	说　明	算术运算符	说　明
+	加号	*	乘号
−	减号	/	除号

例如，

```
InputNumber := 2 + 3;
InputNumber := InputNumber + 1;
```

3. 关系运算符

ASL 中的关系运算符见表 4-8。

表 4-8 关系运算符

关系表达式	说　明	关系表达式	说　明
=	等于	<>、#	不等于
<	小于	<=	小于或等于
>	大于	>=	大于或等于

关系运算符用来构造关系表达式，例如：

```
InputNumber > 1
InputNumber <> 100
```

4. 逻辑运算符

ASL 中的逻辑运算符见表 4-9。

表 4-9 逻辑运算符

逻辑运算符	说　明	逻辑运算符	说　明
AND	与	NOT	非
OR	或		

逻辑运算符用来构造逻辑表达式。例如，表达式 NOT (3 > 2) AND (2 > 1) 的运算结果为“假”。

4.11.2 ASL 中的分支语句

分支语句的格式有两种。

(1) IF <逻辑表达式>THEN 语句。执行 IF 语句时，先计算<逻辑表达式>的值，若为 TRUE，则执行语句，否则不执行任何操作。

(2) IF <逻辑表达式>THEN 语句 1 ELSE 语句 2。执行 IF 语句时，先计算<逻辑

表达式>的值，若为 TRUE，则执行语句 1，否则执行语句 2。

例如，设计 ASL 脚本，判断一个变量是否是正数，如果是，则显示“正数”：

```
VAR a;
BEGIN
  a := 0;
  IF a > 0 THEN
    write('正数');
END.
```

在这段脚本中，应注意以下 7 个问题。

（1）在字符串字面量上使用单引号。下面的脚本中，write(正数)未使用单引号把“正数”引起来，是语法错误的语句。

```
VAR a;
BEGIN
  a := 0; write(a); write('a');
  IF a >= 0 THEN
    write(正数);  //“正数”是字符串字面量,要求用单引号引起来
END.
```

（2）BEGIN 和 END 括起来的语句可以写在同一行上，也可以写在不同行上。语句位置不影响正确性。

```
VAR a;
BEGIN  a := 0;  IF a > 0 THEN write('正数'); END.
```

（3）BEGIN END. 不能省略。下面的写法是错误的。

```
VAR a;
  a := 0;
  IF a > 0 THEN
    write('正数');
```

（4）语句要全部放在 BEGIN END. 中。下面的脚本把“a：=0”语句写在了 BEGIN 和 END 之外，是错误的。

```
VAR a;
a := 0;
BEGIN
  IF a > 0 THEN
    write('正数');
END.
```

（5）大小写不敏感。下面的脚本中，a 和 A 是同一个变量。

```
VAR a;
BEGIN
```

```
  A := 0;
  if a > 0 then
    write('正数');
END.
```

（6）采用适当的缩进，使得源代码看起来清晰。

```
VAR a;
BEGIN
  a := 0;                 //缩进 2个空格
  IF a > 0 THEN           //缩进 2个空格
    write('正数');        //缩进 4个空格
END.                      //与 BEGIN 对齐
```

（7）IF 语句可以写在一行上，也可写在多行上。

```
VAR a;
BEGIN
a := 0;
  IF a > 0 THEN write('正数');
END
```

当有多条语句从属于 IF 语句时，要使用 BEGIN END 把这些语句括起来。

```
VAR a;
BEGIN
  a := -1;
  IF a >= 0 THEN
  BEGIN
    write(a);
    write('是非负数');
  END;
END.
```

再如，设计 ASL 脚本，判断输入的一个数字是否为正数，若为正数，则显示"你输入的数为正数"；若为负数，则显示"你输入的数为负数"；若为 0，则显示"你输入的数是零"。脚本如下。

```
VAR  a;
BEGIN
  Read(a);
IF a <> 0 THEN
  BEGIN
    IF a > 0 THEN
      BEGIN
        Write('你输入的数是正数');
      END
```

```
        ELSE
          BEGIN
            Write('你输入的数是负数');
          END;
    END
  ELSE
    BEGIN
      Write('你输入的数是零');
    END;
END.
```

再看一个例子。判断一个变量 a 是否大于或等于 0，如果是，则显示“是非负数”；否则显示“是负数”。脚本为

```
VAR a;
BEGIN
  a := 1;
  IF a >= 0 THEN
    write('是非负数');
  IF a < 0 THEN
    write('是负数');
END.
```

当变量 a 确实为非负数时，第 2 个分支语句就没有必要执行了。所以，这种情况下最好的脚本应是

```
VAR a;
BEGIN
  a := 1;
  IF a >= 0 THEN
    write('是非负数');
  ELSE
    write('是负数');
END.
```

4.11.3　ASL 中的循环语句

ASL 中的循环语句有两种：当型循环（while 循环）和直到型循环（until 循环）。

while 语句的形式为

```
while <逻辑表达式> do <语句>;
```

其意义为：当逻辑表达式的值为 TRUE 时，执行 do 后面的语句。

while 语句的执行过程为

① 判断逻辑表达式的值，如果其值为真，则执行步骤②，否则执行步骤④。

② 执行循环体语句(do 后面的语句)。

③ 返回步骤①。

④ 结束循环,执行 while 的下一个语句。

这里的 while 和 do 为保留字。while 语句的特点是,当逻辑表达式成立时,重复执行 do 后面的语句(循环体)。

例如,从键盘输入 5 个数:11,2,13,40,10,累加并输出结果。下面的脚本可以完成本例的功能。

```
VAR a,s;
BEGIN
  s := 0;

  Read(a);
  s := s + a;

  Read(a);
  s := s + a;

  Read(a);
  s := s + a;

  Read(a);
  s := s + a;

  Read(a);
  s := s + a;

  Read(a);
  s := s + a;

  Read(a);
  s := s + a;

  Read(a);
  s := s + a;

  Read(a);
  s := s + a;

  Read(a);
  s := s + a;

  write(s);
END.
```

但是，如果题目给出 500 个数，则需要简洁的方法表达重复执行语句“Read(a) 和 s:= s+a”。这就会使用到 while 语句。使用 while 语句的脚本如下。

```
VAR a, s, i;
BEGIN
  s := 0;
  i := 1;
  while  i <= 10 DO
  BEGIN
    Read(a);
    s := s + a;
    i := i + 1;
  END;
  write(s);
END.
```

while 语句的组成部分如图 4-17 所示。

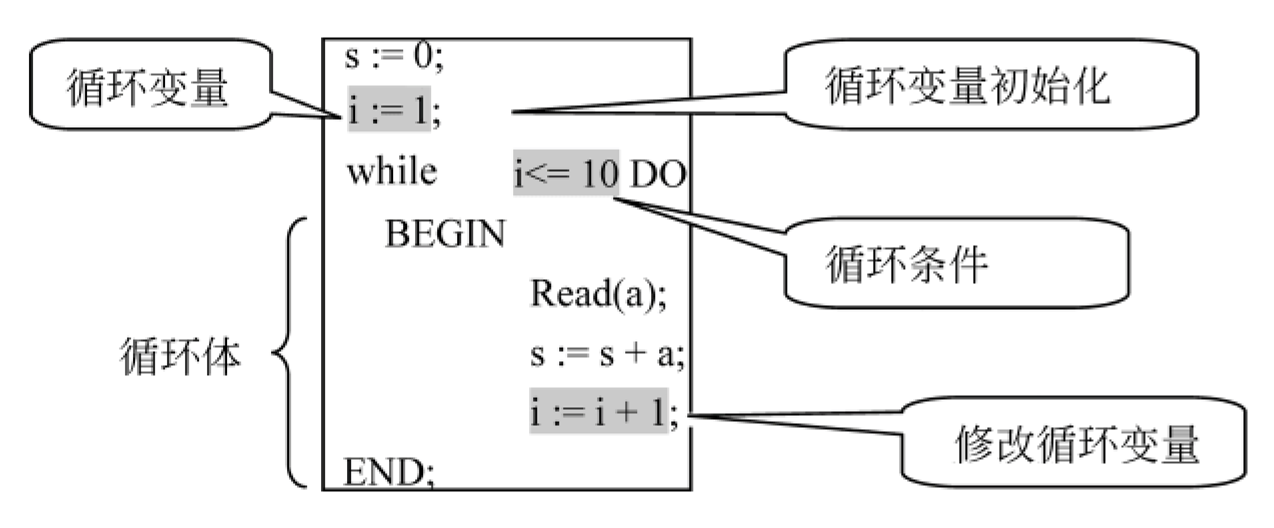


图 4-17　while 语句的组成部分

while 语句的执行流程是：首先判断循环条件是否满足，若满足，则执行循环体，否则执行下一条语句。

使用 while 应注意：循环变量必须初始化；必修修改循环体中循环变量的语句，否则会出现死循环现象。

例如，设计 ASL 脚本，从键盘输入若干个正数进行累加，当输入的数是 0 时结束并输出结果。脚本为

```
VAR a, s;
BEGIN
  // 循环累加正数
  s := 0;
  // 输入一个数字
  Read(a);
  while  a > 0  DO
  BEGIN
    //累加输入的数字
    s :=  s + a;
    // 输入下一个数字
```

```
    Read(a);
  End;
  Write(s); //显示累计结果
End.
```

在这个例子中，常见的错误有 3 个。

错误 1，循环变量没有初始化，如下面的脚本所示。

```
VAR a, s;
BEGIN
  // 循环累加正数
  s :=0;
  while  a > 0  DO      //a 没有初始值,无法判断
  BEGIN
    //累加输入的数字
    s :=  s + a;
    // 输入下一个数字
    Read(a);
  END;
  Write(s);             //显示累计结果
End.
```

错误 2，丢失了边界上的数据，如下面的代码所示。

```
VAR a, s;
BEGIN
  // 循环累加正数
  s :=0;
  // 输入一个数字
  Read(a);
  while  a > 0  DO
  BEGIN
    // 输入下一个数字
    Read(a);            //累加和中丢失了第 1 个数
    //累加输入的数字
    s :=  s + a;
  End;
  Write(s);             //显示累计结果
End.
```

问题 3，忘记使用 BEGIN…END 把循环体括起来，如下面的代码所示。

```
VAR a, s;
BEGIN
  s :=0;
  Read(a);
  while  a > 0  DO
```

```
    s :=  s + a;
    Read(a);
    Write(s);
  End.
```

用 while 语句可以实现“当型循环”；用 repeat-until 语句可以实现“直到型循环”。repeat-until 语句的含义是：“重复执行循环，直到指定的条件为真时为止。”

repeat-until 循环语句的一般形式为

```
repeat
  <语句 1>;
      …
  <语句 n>;
until <逻辑表达式>;
```

其中，repeat、until 是 ASL 保留字，repeat 与 until 之间的所有语句称为循环体。

repeat 语句的特点是：先执行循环，后判断结束条件，因而至少要执行一次循环体。repeat-until 是一个整体，它是一个语句，不要误认为 repeat 是一个语句，until 是另一个语句。repeat 语句在逻辑表达式的值为“真”时不再执行循环体，且循环体可以是若干个语句，不需用 begin 和 end 把它们包起来，repeat 和 until 已经起了 begin 和 end 的作用。while 循环和 repeat 循环是可以互相转换的。

例如，设计 ASL 脚本，提示用户输入一个数字，如果输入的数是 0，则停止；否则将输入的数字累加，并在最后显示累加结果。脚本如下。

```
VAR a, s;
BEGIN
  // 输入一个数字
  Read(a);
  // 循环累加正数
  s := 0;
  repeat
    BEGIN
      //累加输入的数字
      s :=  s + a;
      // 输入下一个数字
      Read(a);
    End;
  until a = 0
  Write(s); //显示累计结果
End.
```

注意，repeat 循环先执行循环体，后判定循环条件；而 while 语句则是先判断循环条件，后执行循环体。所以，repeat 中的循环体至少执行 1 次；而 while 中的循环体可能一次也不执行。

由于 repeat 循环至少执行一次，所以上面的脚本可以在省略循环前读语句，但是需要调换循环体中两个语句的位置。

```
VAR a, s;
BEGIN
  s := 0;
  repeat
  BEGIN
    Read(a);
    s :=   s + a;
  End;
  until a = 0 ;
  Write(s);
End.
```

4.11.4 从脚本中访问数据库

假设已经导入了某财务数据库，其中有表“凭证库”存放了凭证。把 1 月份 1 号凭证和 3 号凭证加入疑点库作为未落实疑点。

首先，在 SQL 查询器中执行以下 SQL 语句，以确认 1 号和 3 号凭证可以从数据库中查询出来。

```
SELECT  DISTINCT  源凭证号
FROM 凭证库 WHERE 会计月份=1 AND (凭证号 = '1' OR 凭证号= '3')
```

当 SQL 语句实验成功后，再执行以下脚本。

```
var Q, E,R,M;
begin
  Q:= createQ('select  DISTINCT  [源凭证号] from [凭证库] where [会计月份]=1 AND ([凭证号] = "1"
OR [凭证号]= "3" ', -1);
  E: = qeof(Q);
  While E <> 1 do
  begin
      R := qfdvalue(Q,'源凭证号');  //字段名“源凭证号”要用单引号引起来
      write(R);//addvourslt(R);
      M := qmov(Q,1);
      E := qeof(Q);
  end;
end.
```

再如，读取凭证库中的一月份凭证，把凭证号为 1 的凭证放在未落实疑点；把凭证号为 2 的凭证放到已落实疑点。脚本如下。

```
VAR Q, E, V, S, M;
BEGIN
```

```
//从凭证库中查询想要的凭证,将查询结果存放到一个地方 (查询变量)
Q := createQ('select * from [凭证库] where [会计月份]=1 AND ([凭证号] = "1" OR [凭证号]= "2")',-1);
  E := qeof(a);
  While E <> 1 do
  begin
      //查看当前行的凭证号是否是 1,如果是,则放入未落实疑点
      V := qfdvalue(a,'凭证号');
      IF V = 1 THEN
        BEGIN
           S := qfdvalue(a,'源凭证号');
           addvourslt(S);
        END;
      //查看当前行的凭证号是否是 2,如果是,则放入已落实疑点

      IF V = 2 THEN
        BEGIN
           S := qfdvalue(a,'源凭证号');
           addvouDout(S);
        END;
      M := qmov(Q,1);
      E := qeof(Q);
  end;
END.
```

注意：如果同一次重复存入相同的凭证（以源凭证号标识），则在疑点库中只保存一个。

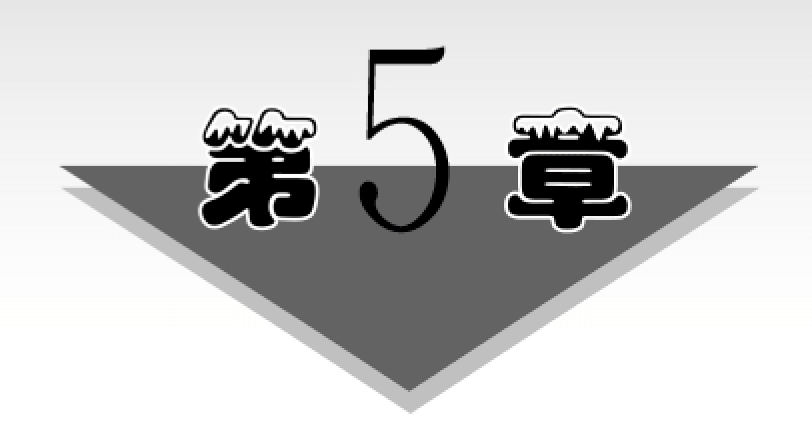
ASL 中相关访问数据库的函数的解释见表 4-10。

表 4-10　ASL 中相关访问数据库的函数的解释

函　　数	功 能 说 明
CreateQ('SQL 语句',查询变量)	执行 SQL 语句，返回一个 SQL 语句的执行及结果到查询变量中。第一个参数是查询的 SQL 语句，第二个参数是使用过已经没有用的查询变量，如果是第一次作查询，则填写－1，这样做的好处是，节省查询变量的个数，节约资源
QEof(查询变量)	查看查询变量当前指向的记录是否为空。若返回 1，则表明为空
QMov(查询变量，移动方向)	在查询变量中存有多条记录，其中有一条为当前记录。该函数可把当前记录的前一条或后一条记录设置为新的当前记录
QFDValue(查询变量，'字段名称')	取当前查询变量当前记录的字段的值。入口参数一个是查询变量，一个是某一个字段的名称，需要用单引号括起来
RecordNum(查询变量或表名称)	如果参数为一个打开的查询变量，则取出当前查询变量的记录个数；如果参数为数据库中的一个表名，则取出该表的所有记录个数
ExecuteUpdate('SQL 语句')	执行数据库插入、更新、删除(Insert，Update，Delete)的 SQL 语句的函数

续表

函　　数	功能说明
CreateTempTable('新表名','SQL 语句')	把 SQL 语句取出来的结果以用户输入的表名存储在数据库中
AddVouDout(源凭证号)	输出源凭证号到疑点库中,该源凭证号对应的凭证将被定义为已落实疑点
AddVouRslt(源凭证号)	输出源凭证号到疑点库中,该源凭证号对应的凭证将被定义为未落实疑点
AddTransDout(查询变量,'疑点说明')	把指定查询变量的当前记录输出到业务疑点的临时库中
AddTransRslt(查询变量,'疑点说明');	把指定查询变量的当前记录输出到业务结果的临时库中
TransBatch(查询变量,'批量疑点说明')	把指定查询变量的业务疑点或结果临时库中的数据写入业务疑点或结果库中,并清空相应的临时库中的数据
showmsg('消息内容')	显示消息



多维数据分析技术

多维数据分析基于多维数据库观察数据间关联和趋势。多维数据库以事实、维度或者度量值对数据进行建模。分析服务是对多维数据库进行数据管理和分析的引擎。其体系结构如图 5-1 所示。该体系结构的核心是分析服务，如 Microsoft SQL Server OLAP Service。来自财务库和业务库的会计事务和业务事务数据经过数据清洗、转换和验证，形成新的数据库。基于新的数据库，建立多维分析模型。通过分析服务运行多维分析模型，通过数据透视表等工具浏览和分析，发现审计线索。应用多维分析的优势在于能够直观地把握总体、分析趋势、发现异常、选择重点。

一个多维数据库中可以有多个多维数据集，通过分析服务访问多维数据集。通过 Microsoft Excel 能够观察透视图和钻取，分析趋势和发现异常更加方便。

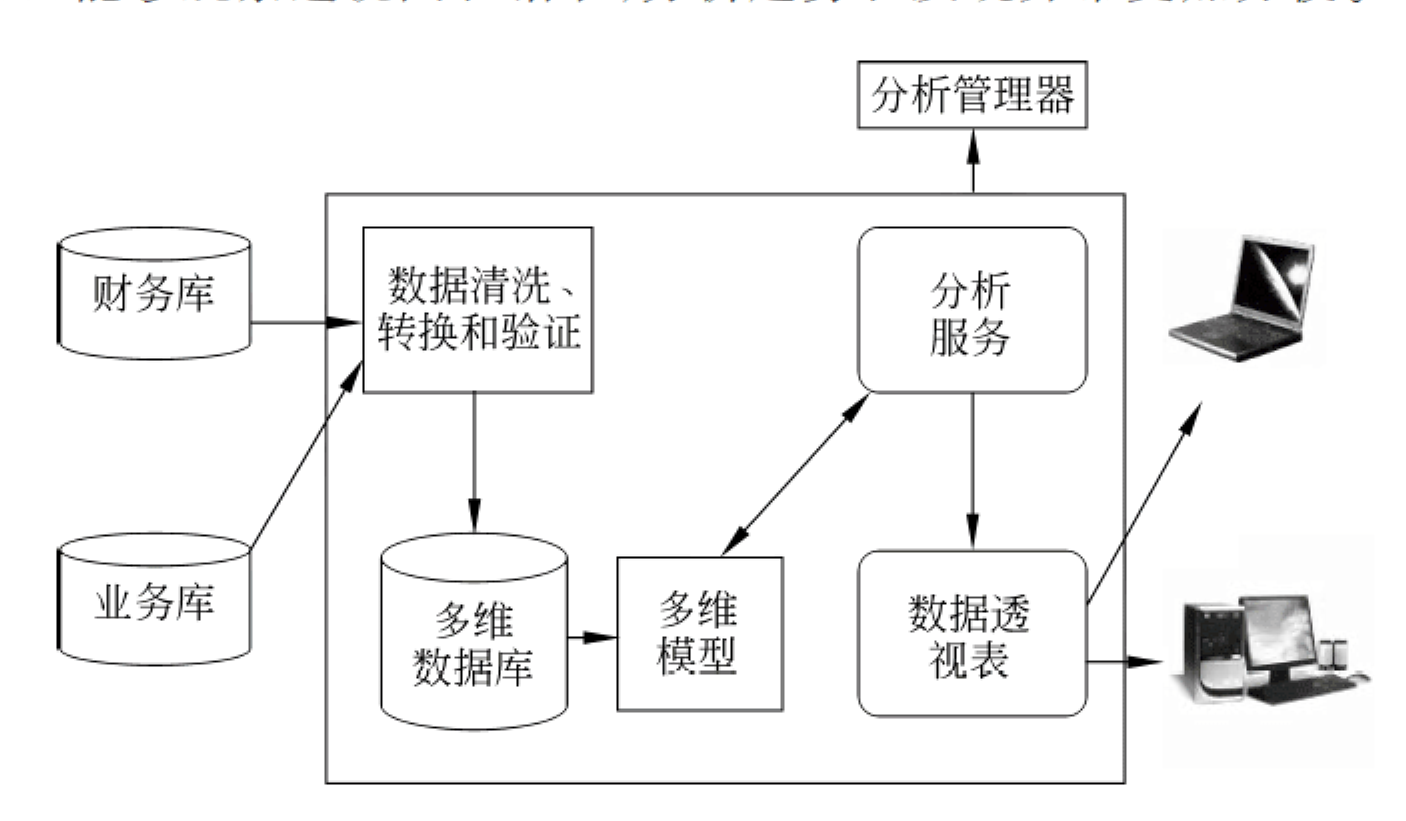


图 5-1　分析服务的体系结构

建立多维分析模型的关键是根据业务需要，设计维和度量值。凡是能够对数据进行分组的实体，如部门，就可以作为**维**。事实表中的数值列可以按照某个维进行分组汇总，这样的列称为**度量值**，来自同一个事实表的若干度量值形成一个**度量值组**。对事实的计数也可以作为度量值。

多维数据库的数据源通常是关系型数据库。该数据库中的数据模式(库结构)往往被转换成 OLAP 多维数据库中的星型或者雪花型模式。只要能通过 ODBC 或者 OLE DB 连接，多维数据库中的数据也可以是非关系型数据库。一般情况下，OLE DB 和 ODBC Provider 作为数据访问组件的一部分与 SQL Server 一起安装。

分析服务是对多维数据库进行数据管理和分析的引擎。审计人员往往需要一些统计

汇总信息，而在大规模关系数据库中通过 SQL 语句进行统计汇总是一个复杂而且耗时的工作。为了解决这个问题，需要建立一个独立的数据库更好地存储用来进行分析的业务数据。这个数据库的数据来源于操作型数据库，但必须转换成星型或者雪花型。在这种架构下，通常有一个事实表。事实表记录了业务活动数据，如延期纳税审批活动。这张表会有上万，甚至几十万行数据。维表定义了维的属性、级别和各级别的成员。

典型的星形模式有以下特点：只有一个事实表，事实表中含有数值列和外键列。每个维只有一个维表，表中包含了该维的主键及其他属性。主键用来作为事实表的参照，其他属性一般是对主键的进一步分组。例如，“产品”维一般有“品牌”属性，“品牌”也是对产品的一种分组。

雪花架构将一个维拆成若干个维。可将维表中成员之间的多对一关系定义为一个独立的有层次体系的维表。某一维的分离使得整个数据模型看起来像雪花，因此称为雪花模型。

分析服务对数据的一致性十分敏感。当定义好多维模型，分析服务就要从关系数据库中读入数据，按照多维数据集结构的要求进行分组汇总，此时需要数据满足以下一致性规则：无空值(null)和无效值(invalid)、参照完整、不允许删除事实表中的行。

维有 3 种类型：常规维、父子维和具有层次结构的维。表示常规维的维表通常有偶数个编码和名称相对应的列。父子维与常规维不同的是，其维表只有 3 列，其中两列表达了父子关系。例如，会计科目中包含 3 列：科目编码、科目名称、上级科目编码。其中，科目编码和上级科目编码两列中包含了父子关系。

多维数据模型对数据的要求有

(1) 度量值必须是数值型。因为只有数值型列，才能被计算汇总。在 Access 中，数值型数据类型有数字型和货币型。SQL Server 中的数值型数据有 bigint、int、smallint、tinyint、numeric、decimal、smallmoney、money、real 和 float。

(2) 事实表和维度表间具有参照关系的列必须为相同数据类型。

(3) 维成员是字符型。

(4) 时间维也是常规维，其对应的列必须是日期时间型。Access 中的列数据类型选“日期/时间”。SQL Server 中的列数据类型选择 smalldatetime 或者 datetime。

5.1 多维分析案例——延期纳税

按照税收征管法及实施细则规定，只有当企业出现以下情况时，税务机关才能批准企业延期纳税：因不可抗力，导致纳税人发生较大损失，正常生产经营活动受到较大影响的；当期货币资金在扣除应付职工工资、社会保险费后，不足以缴纳税款的。本案例的审计目标就是查证有无非法批准企业延期纳税。

审前调查得知，某市国税局所属各区县的延期纳税审批数据全部集中在市局的数据库服务器中。通过终端查询和后台转换采集了数据，并进行了代码转换，最后将其导入 Access 数据库中：X 市国税局批准延期纳税.mdb(图 5-2)。其中有 3 个表：延期纳税批件、税务机关代码和征收项目代码。3 个表的表结构及代表性或全部数据如下。

延期纳税批件表：

字段名称	数据类型	说明
纳税人识别号	文本	
征收项目代码	文本	
所属期起	日期/时间	
所属期止	日期/时间	
税额	数字	
税务机关代码	文本	
审批日期	日期/时间	
纳税人名称	文本	

税务机关代码表：

字段名称	数据类型	说明
税务机关代码	文本	
税务机关名称	文本	

征收项目代码表：

字段名称	数据类型	说明
征收项目代码	文本	
征收项目名称	文本	

图 5-2　X 市国税局批准延期纳税数据库

审计人员希望以图 5-3 所示方式在不同维上浏览数据：指定所有或某个征收项目，列出各区(县)税务机关在各年度或者某年度各月的审批延期纳税额情况，并对可疑数据项可以立即调出其在延期纳税批件表中的相关行。

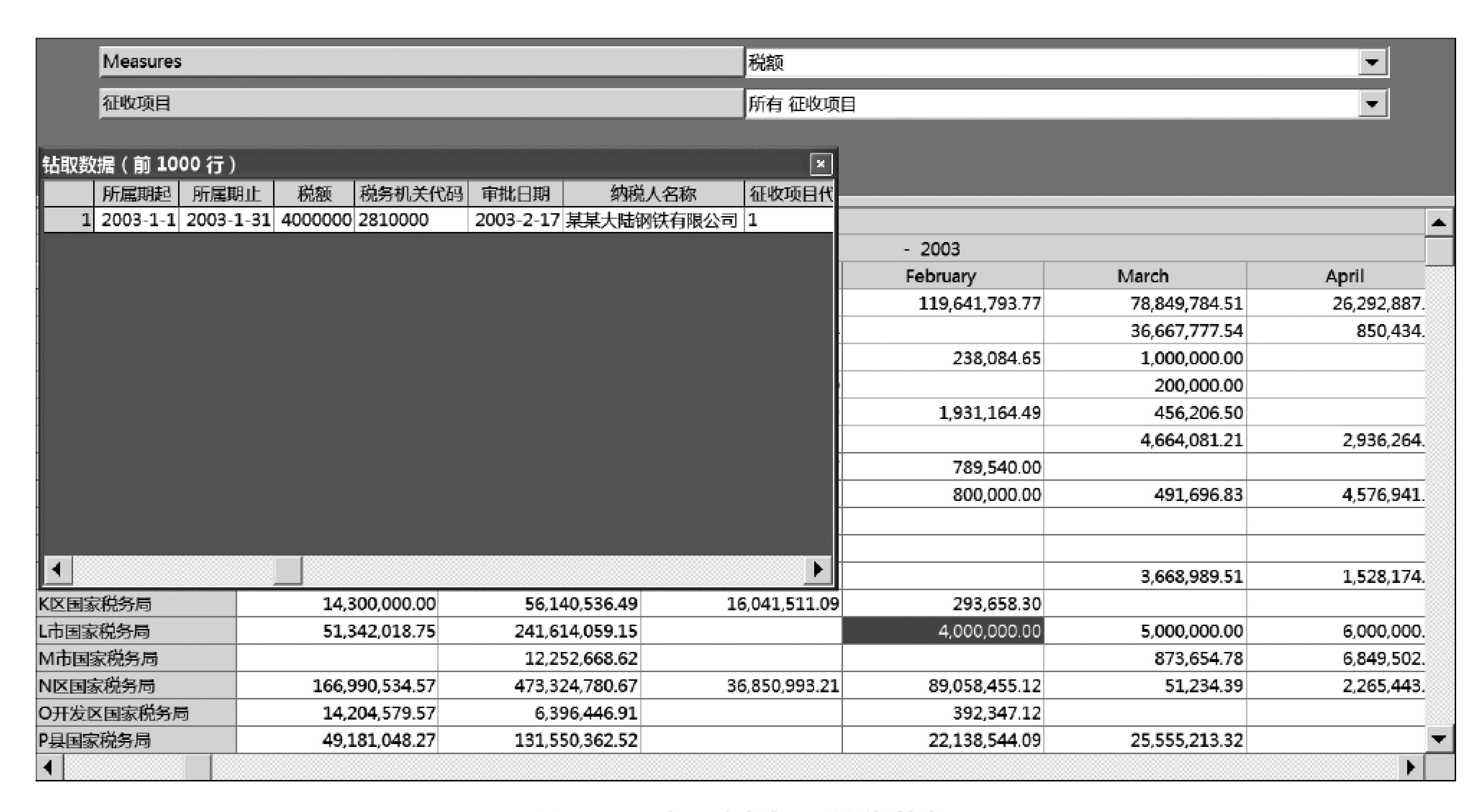


图 5-3　在不同维上浏览数据

以 Microsoft Visual Studio 为开发工具，以 Microsoft Analysis Services 为分析服务，以 Microsoft Excel 为客户端工具，建立和浏览多维分析模型的过程如下。

首先以"Microsoft Jet 4.0 OLE DB Provider"作为驱动程序定义分析服务与"X 市国税局批准延期纳税.mdb"的数据源。其中，延期纳税批件为事实表，征收项目代码和税务机关代码为维表。

然后定义事实表和维表之间的参照关系如图 5-4 所示。多维分析模型中的各个表必须定义主键。如果在数据源中没有定义,则需定义逻辑主键。

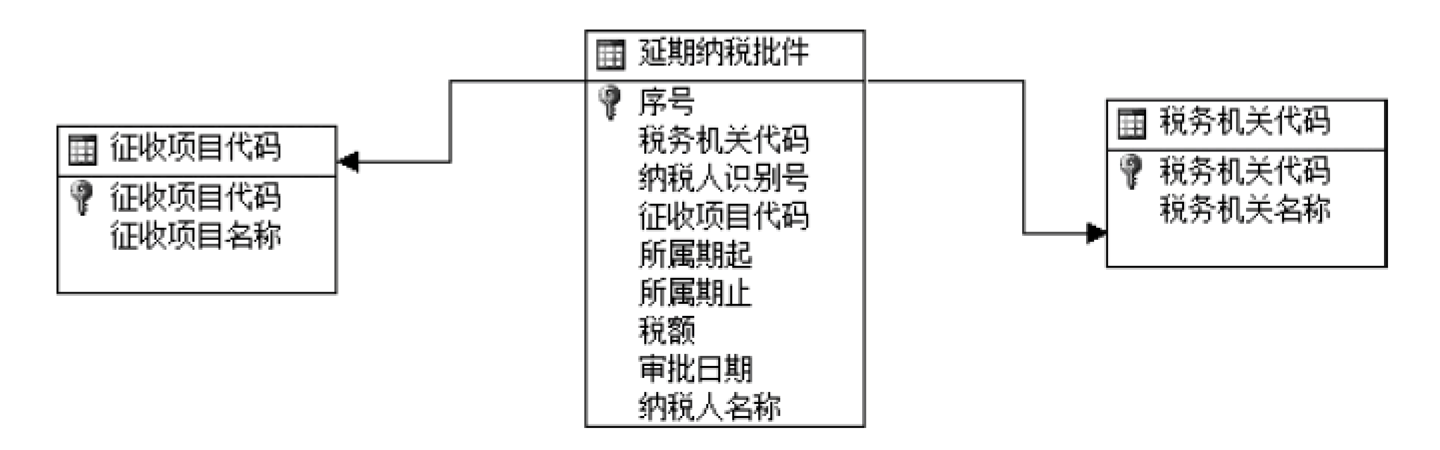


图 5-4 事实表和维表之间的参照关系

接下来设置"延期纳税批件"为事实表,并把其中的"税额"设置为多维数据集中的一个度量值。事实表参照的外键表作为维表。多维数据模型如图 5-5 所示。该模式是一个星型架构。标题条是黄色的表是事实表;标题条是蓝色的表是维表。

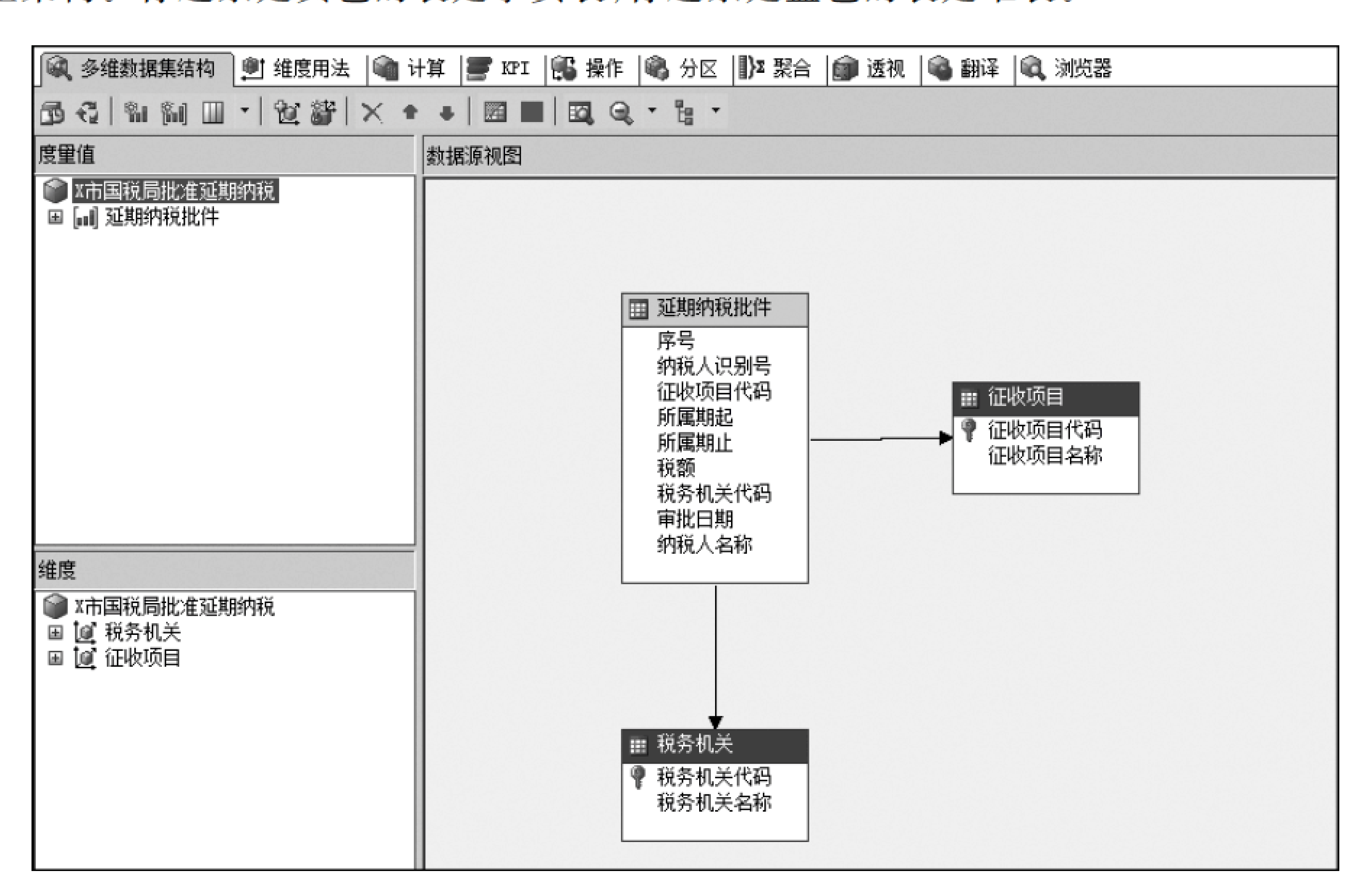


图 5-5 多维数据模型

让分析服务处理该多维数据模型后,就可以按照图 5-3 所示形式浏览数据了。增加"审批日期"维,可按照"审批日期"重新进行处理和浏览。若要通过"审批日期"实现按年、按月、按日期进行数据浏览,则要创建维的级别实现此目标。在维上可以定义多个级别,如"审批日期"维的级别可以有年、季度、月份、星期、日期等。每个级别一般有若干成员,如"星期"级别中的成员有星期一、星期二、星期三、星期四、星期五、星期六和星期日,"月份"级别中的成员有 1 月、2 月、3 月、4 月、5 月、6 月、7 月、8 月、9 月、10 月、11 月、12 月;"季度"级别中的成员有 1 季度、2 季度、3 季度、4 季度。通常,维的所有成员组成了一个层次结构。图 5-6 以时间维为例展示了维的级别以及各级别的成员(矩形框)。

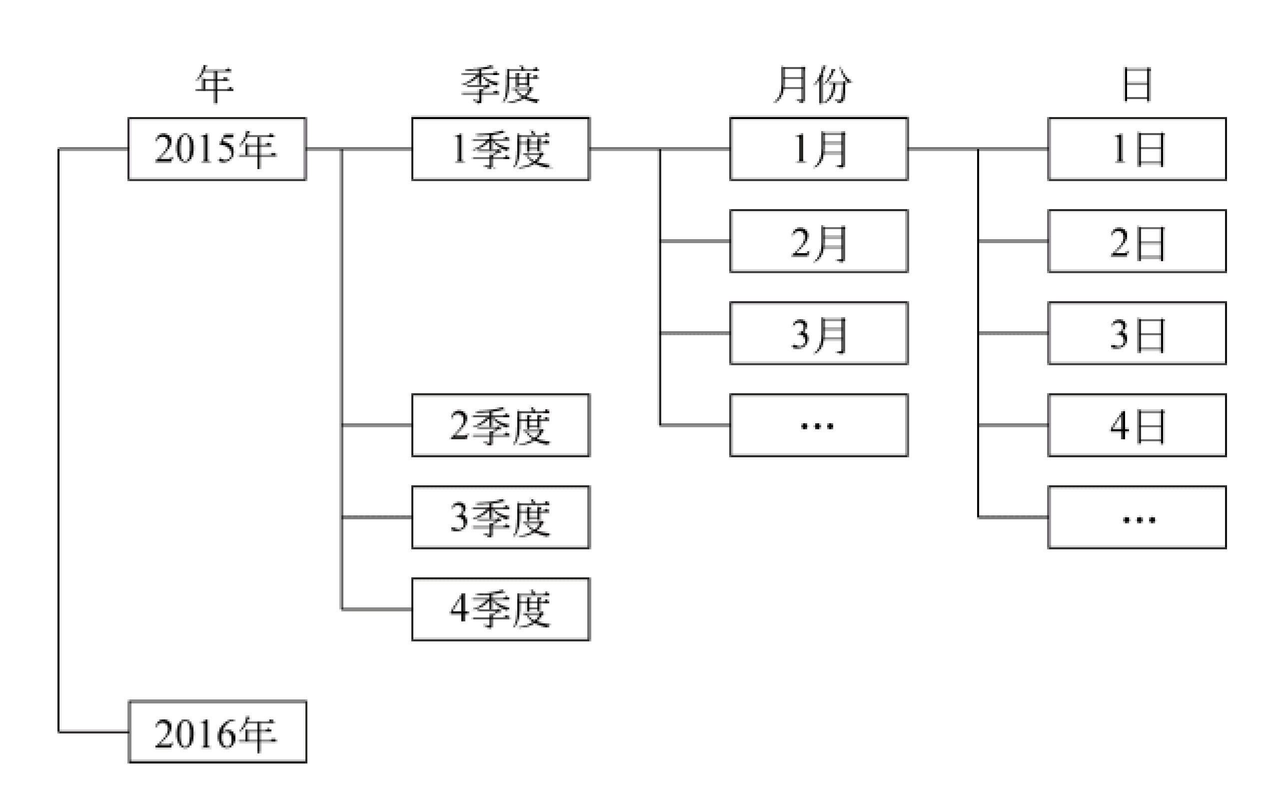


图 5-6　级别

在多维数据集浏览器窗口可观察指定所有或某个税务机关、指定所有或某个征收项目在各个时间维级别（年、月）上的税额（度量值，MeasuresLevel）情况。例如，双击 2012 年，就展开了该年的下级维度（月）。

又如，从“征收项目”维的下拉列表中选择“企业所得税”，可以发现 2012 年和 2014 年所有税务机关没有征收企业所得税，2013 年的企业所得税征收集中在 4 月份和 10 月份。由此可见，通过多维观察数据，审计人员可以很容易发现一些线索。例如，可以看到不少税额为整数，而纳税人应该缴纳的税额很少出现整数，尤其 L 市国家税务局从 2013 年 2 月起连续出现审批金额为整数（35400000）的情况，这是一条重要审计线索。

增加“纳税人”维度，并通过“钻取”操作可以发现，L 市国税局连续对“某某大陆钢铁有限公司”进行批缓。采取同样的方法，还可以发现其他涉嫌企业。通过延伸审计，很容易确认国税局的审批行为是否恰当。

通过 Excel 数据透视表从不同的角度查看数据。数据透视表的名字来源于它具有“透视”数据的能力，这使其成为非常强大的数据分析工具。通过 Excel 浏览延期纳税多维数据集的步骤如下。

创建 Excel 数据透视表。单击 Excel“数据”菜单，选择“自其他来源”命令，然后选择“来自分析服务”，设置分析服务器名称为“.”，表示本地服务器。在“选择数据库和表”对话框中选择数据库“延期纳税”。设置 Excel 进行数据库连接的文件名字，单击“完成”按钮，此时 Excel 如图 5-7 所示。

从图 5-7 中可以看到，“数据透视表字段列表”中列出了多维数据集中的 3 个维度（审批日期、税务机关和征收项目）和一个度量值（税额）。通过把维度拖放到下方的 3 个不同区域（行标签、列标签和报表筛选）建立透视表。

Excel 的优势在于，它可以将数据透视表和数据透视图联动展示。在 Excel 中选择一个新的工作单（sheet）或者新建一个工作单。在“数据”功能区中选择“现有连接”。从中选择先前建立的“X 市国税局批准延期纳税”，在导入数据对话框中选择“数据透视图和数据透视表”，单击“确定”按钮后，在 Excel 工作单上既出现了透视表区域（左上角），又出现了透视图区域（中间空白处）。“数据透视表字段列表”的上半部分列出了来自多维数据集的度量值和维度；其下半部分是数据透视表或者数据透视图的参数设置，如图 5-8 所示。

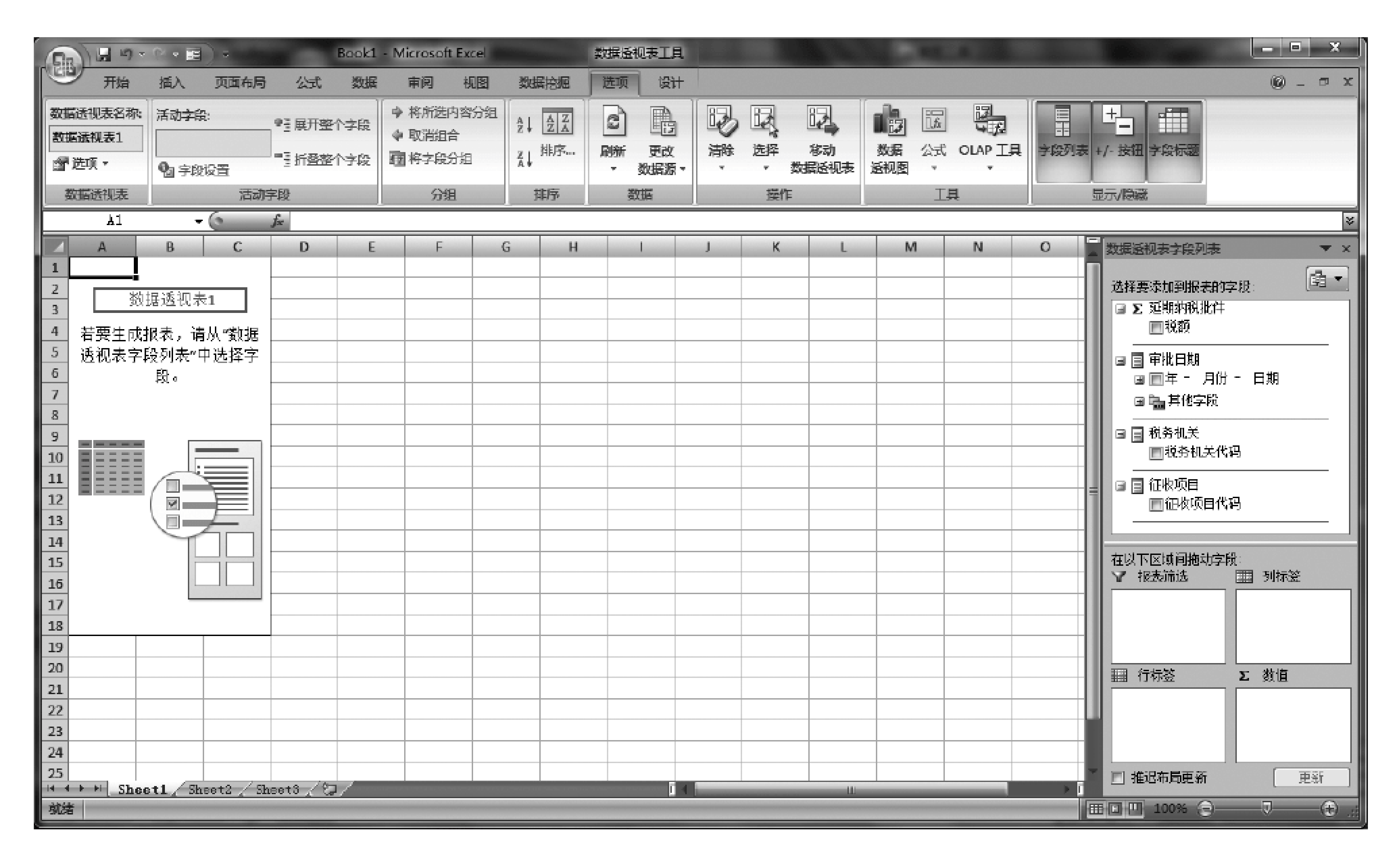


图 5-7　初始的数据透视表

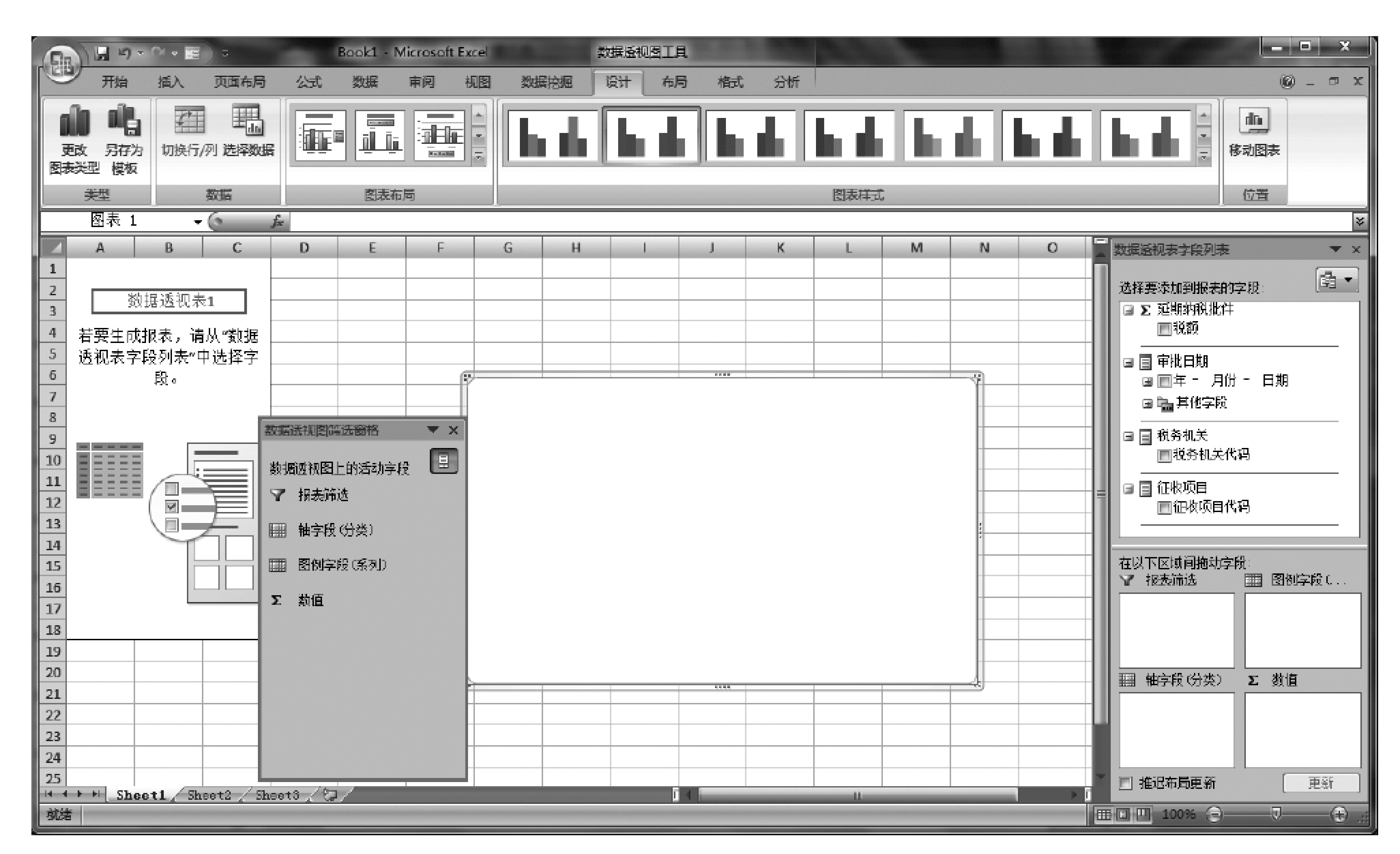


图 5-8　数据透视图

如果单击数据透视图区域，则显示“数据透视图筛选窗格”，窗格中包括“报表筛选”“轴字段(分类)”“图例字段(系列)”和“Σ数值”；如果单击数据透视表，则相应“轴字段(分类)”变成了“行标签”，相应“图例字段(系列)”变成了“列标签”。这个对应变化反映了数据透视图和数据透视表间的一致性：数据透视表中的行标签对应于数据透视图的轴字

段；数据透视表的列标签对应于数据透视图的图例字段。

在“选择要添加到报表的字段”中选择度量值“税额”，该度量值就出现在“∑数值”框中，意思是在图中显示对该度量值的各种汇总。如果想按2013年度观察各税务机关批缓情况，就把“审批日期”拖放到“轴字段(分类)”，把“税务机关”拖放到“图例字段”。右击数据透视图的空白区域，可选择图表类型，如条形图、折线图等。

5.2 多维数据集的设计

多维数据集(CUBE)是数据源、数据源视图、维度、度量值和计算列的总称。多维数据集的模型设计一般分4个步骤：设计概念模型、设计度量值、设计维度、设计事实表和维表。

概念模型即期望展现的业务目标，如各个税务机关延期纳税审批合法情况，这就是一个目标。目标要体现出某一方面的各分析角度(维)和统计数值型数据(度量值)之间的关系。例如，从日期、税务机关、征收项目3个角度观察延期纳税审批金额。

建立多维模型的关键是度量值的选取、维的选取、维级别设计、维成员属性的设计。

从概念模型中确定度量值，度量值是可以被分组汇总的数值列，如审批金额。在确定了度量值之后，审计人员要考虑不同维度下度量值的聚合情况，通常采用“最小粒度原则”，即将度量值的粒度设置到最小。

例如，假设目前的延期纳税数据最小记录到日，即数据库中记录了每一天的延期纳税审批业务。如果审计人员在未来的数据分析中只需要精确到月，那么在数据采集转换时按月形成事实表的行，此时，多维数据集中度量值的粒度就是“月”。如果不能确认将来的分析需求在时间上是否需要精确到月，那么就需要遵循“最小粒度原则”，在事实表中保留每一天的数据，以便日后对“天”进行分析。

维是指数据分析的各个角度。能够作为维度的列具有对度量值的分组汇总能力。也就是说，这些列可放在SQL查询语句的ORDER BY子句中。例如，希望按照日期，或者按照税务机关，或者按照征收项目进行分析，那么这里的日期、税务机关、征收项目就是相应的维。基于不同的维，可以看到各度量值的汇总情况，也可以基于所有的维进行交叉分析。

这里首先要确定维度的层次(hierarchy)和级别(level)。例如，在时间维度上，按照“年-季度-月”形成了一个层次，其中“年”“季度”“月”成为这个层次的3个级别；同理，当建立征收项目维度时，可以将“征收项目大类-征收项目子类-征收项目”划为一个层次，其中包含“征收项目大类”“征收项目子类”“征收项目”3个级别。

将3个级别分别对应维表中的3个列。也可以使用3张表，分别保存征收项目大类、征收项目子类、征收项目3部分数据，此时便于各级别属性设计。

被审单位提供了大量数据，里面是一笔笔凭证记录、一笔笔审批记录，等等。那么，这些记录是审计人员将要建立的事实表的原始数据，即关于某一主题的事实记录表。

事实表和维表是多维模型中的两个基本概念。事实表是数据分析对应的主要数据项，一般是企业内的某项业务或某个事件，如延期纳税批件。事实表中的事实一般具有数

据特性和可加性,这种特征对于审计分析而言是非常重要的。在审计分析中,审计人员关心的通常不是单一的一行数据,而是按不同角度和粒度的汇总数据。度量值是所分析的多维数据集的中心值,是基于多维数据集的事实表中的一列。

维表中包含的一般是描述性的文本信息,这些文本信息将成为事实表的分组条件,如按税务机关分类查询延期纳税审批金额,或按月观察延期纳税金额变化趋势等。维表中的维属性应该具体明确,体现出维层次的划分,能够成为分析型查询的约束条件。

在多维模型中,事实表的行称为事实,一般含有数值列或其他可以进行计算的数据。

事实表中除了各维外键和各度量数据,不应该存在冗余描述性信息,即符合"瘦高原则",即要求事实表数据行数尽量多(粒度最小),而描述性信息尽量少。

多维数据模型一般含有一个事实表和一组维表,形成最常见的星形模式。在星形模式中,事实表居中,多个维表呈辐射状分布于其四周,并与事实表连接。

星形模式虽然是一个关系模型,但是它不是一个规范化的模型。在星形模式中,维度表被故意地非规范化了,这是星形模式与 OLTP 系统中的关系模式的基本区别。

采用星形模式设计的数据仓库的优点是,由于数据的组织已经过预处理,所以查询访问效率较高。对于非计算机专业的用户而言,星形模式比较直观,通过分析星形模式,很容易进行各种分析。

在实际应用中,随着事实表和维表的增加和变化,星形模式会产生多种衍生模式,其中包括雪花模式。雪花模式是对星形模式维表的进一步层次化,将某些维表扩展成事实表。雪花模式的维表是基于范式理论的,通常部分表满足第三范式。在某些情况下,雪花模式的形成是由于星形模式在组织数据时,为减少维表层次和处理多对多关系而对数据表进行规范化处理后形成的。

多维模型允许审计人员方便地从各个角度观察数据,而不必设计 SQL 语句组合不同的条件去查询。例如,"延期纳税"案例中的多维模型如图 5-9 所示。

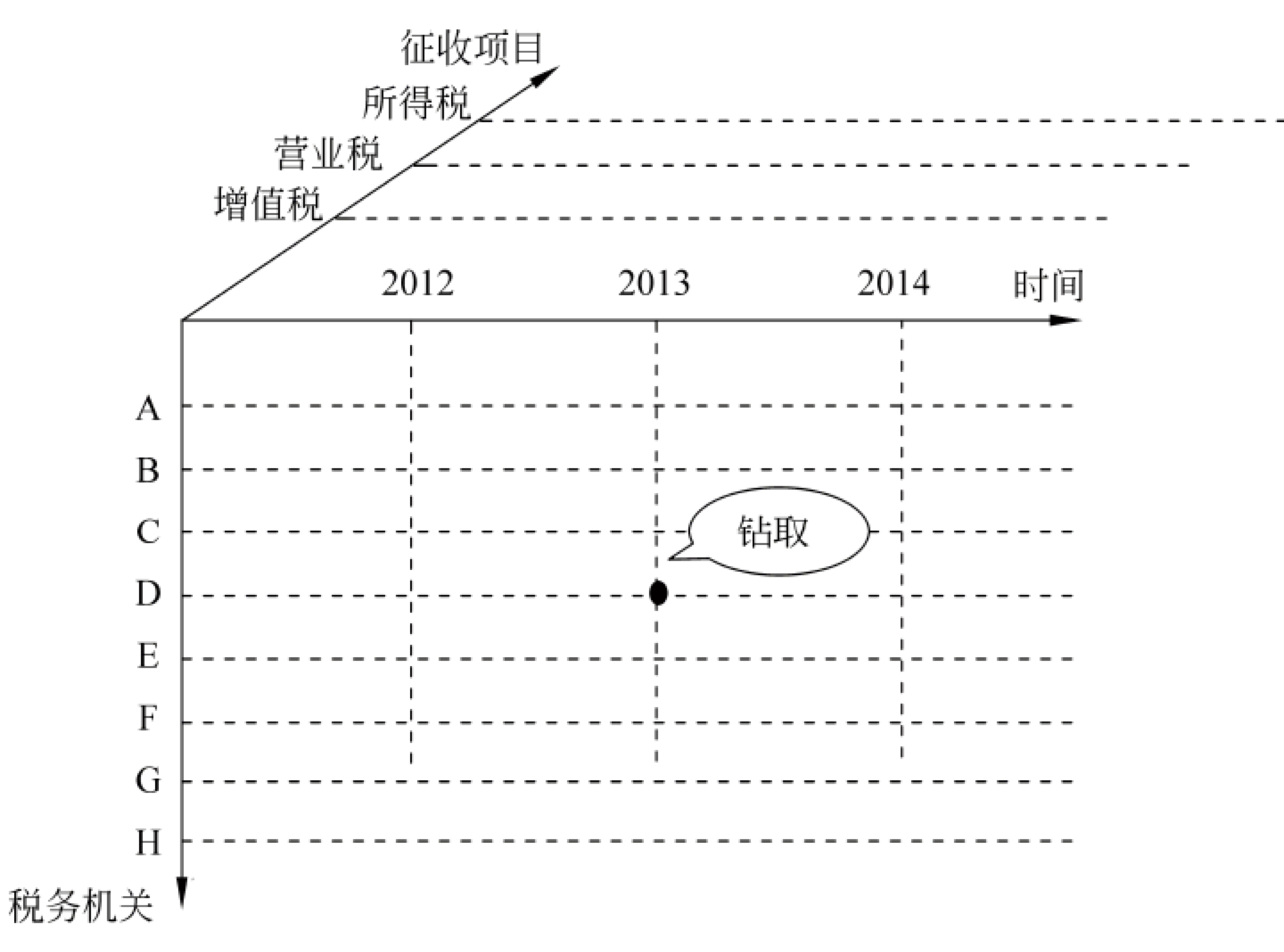


图 5-9 "延期纳税"案例中的多维模型

该模型中被观察的数据是税额（虚线交点上），这些数据被组织到三维空间：税务机关、时间和征收项目。选定某个征收项目，在平面上就呈现出该征收项目下各个税收机关在各个时间（年月）点上的税额情况。对疑点税额，可以通过钻取提取底层明细数据。

分析服务并不验证参照完整性。如果多维数据集的事实表含有不存在的外键，那么对这些行不予处理。数据的不完整，可能导致分析结果错误。

例如，“延期纳税”案例中有一个事实表（延期纳税批件）和一个维表（征收项目代码）。其中，“延期纳税批件”表中的“征收项目代码”列引用了“征收项目代码”表中的“征收项目代码”。假设“延期纳税批件”表中存在征收项目代码为 404 和 410 的行，而在“征收项目代码”表中并没有代码为 404 或 410 的行。这种情况称为引用不完整。如果验证数据时发现有这类问题，就需要对不完整的数据进行补充。

可以采用如下的 SQL 语句进行引用完整性验证。

```
SELELCT DISTINCT 征收项目代码
FROM 延期纳税批件
WHERE 征收项目代码 NOT IN(SELECT 征收项目代码 FROM 征收项目代码)
```

一般来说，最好在建立多维数据集前，通过主键约束和外键约束先建立事实表和维表之间的参照完整性约束。一旦建立了参照完整性约束，SQL Server 就会自动发现违反约束的行，并提出警告。

只有有针对性地采集数据，根据多维数据集对数据的要求进行正确地清洗和转换，并验证数据的真实性和完整性，才能顺利地进行多维分析，发现审计线索和证据。

5.3 多维分析案例——烟草公司纳税

假设业务目标是确认 A 市烟草公司销售额与纳税的一致性。

审计人员需要观察 A 市及下属各县级烟草公司各日期各品牌的销售数量、销售额、税额。据此目标，首先确定度量值。此主题中度量值涉及 3 个：销售数量、销售额、税额。接下来确定事实数据的粒度。从被审单位了解到，形成销售事实的操作型业务数据以天为单位存入数据库，即销售日期精确到天。按照“最小粒度原则”保持此粒度的数据。由于审计数据分析主题中要求从各县级烟草公司、各日期、各品牌进行分析，所以烟草公司、时间和品牌自然形成了待构建的多维数据集的维。通过上面的分析，已经初步形成了星形模式的多维数据集概念模型，如图 5-10 所示。

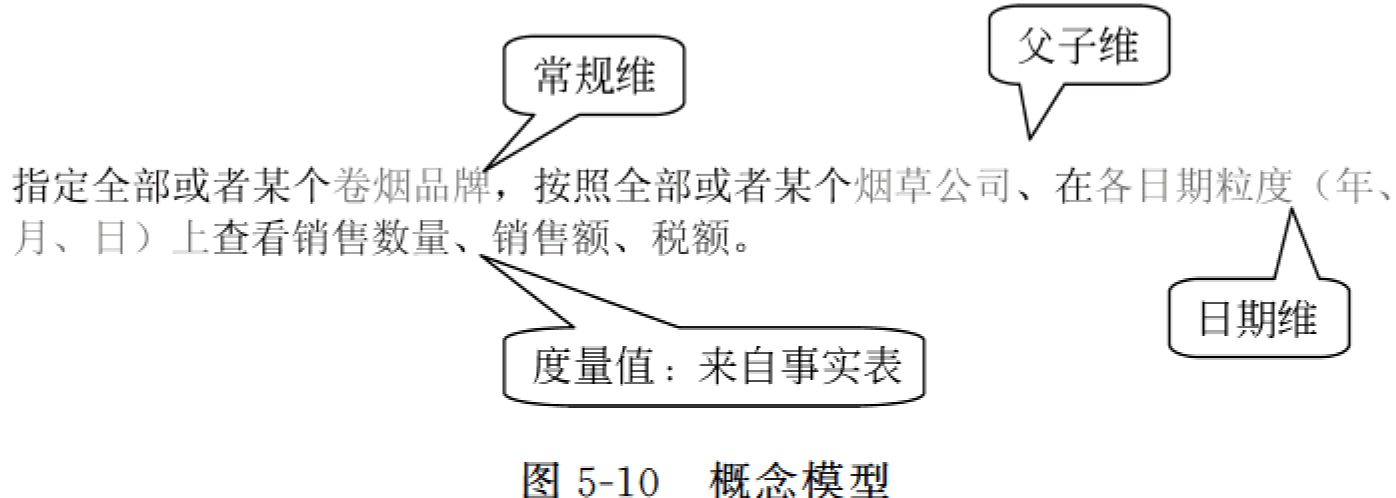


图 5-10　概念模型

5.3.1　创建多维数据集

接下来根据被审单位采集电子数据，并经过清洗、转换和验证后形成一个事实表和两个维表，各表的结构如图 5-11 所示。

销售事实表

字段名称	数据类型
公司ID	文本
部门ID	文本
销售单标识	文本
客户标识	文本
卷烟代码	文本
计量单位	文本
销售数量(条)	数字
销售金额(价税合计)	数字
销售税额	数字
销售日期	日期/时间

烟草公司表

字段名称	数据类型
公司ID	文本
公司名称	文本
上级公司ID	文本
省ID	文本
地区ID	文本

卷烟品牌表

字段名称	数据类型
卷烟代码	文本
卷烟名称	文本

图 5-11　表结构

第一步，启动 Visual Studio，新建一个多维分析项目，命名为“A 市烟草公司”。

第二步，新建数据源。在“解决方案资源管理器”中右击“数据源”，选择“新建数据源”。在“选择如何定义连接”对话框中单击“新建”。在“连接管理器”对话框中首先选择“提供程序”，在其右侧的下拉列表中选择“Microsoft Jet 4.0 OLE DB Provider”，这是 Access 的驱动程序。单击“浏览”按钮，设置存放源数据的 Access 数据库文件位置。单击“确定”按钮，回到“如何定义连接”对话框。设置以“继承”方式登录，最后把数据源命名为“A 市烟草公司”。

第三步，新建数据源视图。在“解决方案资源管理器”中右击“数据源视图”，选择“新建”，在数据源视图向导的“选择数据源”对话框，保持默认的关系数据源。单击“下一步”按钮。在“名称匹配”对话框保持默认设置。在“选择表和视图”，单击“>>”按钮，选择“可用对象”列表中的所有表到右侧列表。

最后把数据源视图命名为“A 市烟草公司销售数据”。在自动打开的数据源视图设计器窗口定义逻辑主键和表间关系，最后的结果如图 5-12 所示。注意，一定通过把“上级公司 ID”拖放到“公司 ID”建立烟草公司间的自引用关系。这个关系是多维数据集向导生成父子维度的关键信息。从概念上看，部门之间的从属关系形成了父子关系，一个成员具有父子关系的维度能够帮助观察者轻松地逐级展开进行观察。所以，一个维度，如果其成

员具有父子关系性质，就适合作为父子维。



图 5-12　数据源视图

第四步，新建多维数据集。右击“多维数据集”文件夹，选择“新建”，向导通过数据源链接从 Access 数据库 A 市烟草公司销售数据. mdb 中读取出了 3 个表：烟草公司表、卷烟品牌表和销售事实数据表。选择“销售事实数据表”作为本多维数据集的度量值组表。下一步要指定在事实表中哪些数字列用来作为度量值。向导从数据库中找到了 3 个数字列：销售数量(条)、销售金额(价税合计)和销售税额。所有数字列作为度量值。下一步定义多维数据集的维：在“选择新维度”对话框保持默认设置。最后把多维数据集命名为“A 市烟草公司销售数据”。

第五步，编辑生成的维度。在“解决方案资源管理器”中双击“烟草公司”维度，打开维度设计器，在“维度结构”标签页单击“公司 ID”，在其属性窗口设置 NameColumn 属性为“公司名称”。

A 市烟草公司下属 18 个县级销售公司，县级公司直接面对零售商。各单位间从属关系可通过公司代码表中公司 ID 列和上级公司 ID 列获取。这种存在父子关系的维通过在数据源视图定义的自引用关系被向导自动识别，创建为父子维。单击“烟草公司”维的“上级公司 ID”属性，可以在其属性窗口中看到其用法属性(Usage)为 Parent。采取同样的步骤，设置“卷烟代码表”维的 NameColumn 的属性为“卷烟名称”。

第六步，新建日期维度。在多维数据集设计器窗口右击“销售事实数据表”，选择“浏览”，观察到日期列从 2003 年 1 月到 2003 年 12 月。所以，日期维度的设置如下：第一个日历日为 2003 年 1 月 1 日；最后一个日历日为 2003 年 12 月 31 日；时间段为年、月份和日期。最后指定维的名字：销售日期。单击“完成”按钮，然后在多维数据集设计器中把该维度添加进来。

第七步，编辑“维度用法”。在多维数据集设计器中选择“维度用法”标签页，保持默认的公司代码和卷烟代码维度关系不变，定义销售日期维度和度量值组列的关系。在解决方案资源管理器中展开 A 市烟草公司的数据源文件夹、多维数据集文件夹和共享维度文件夹，可以发现已经创建了一个数据源、一个多维数据集和 3 个维。单击工具栏中的“全部保存”按钮。

第八步，右击销售多维数据集，选择“浏览数据”。此时，审计人员就可以从销售公司、销售日期和卷烟名称 3 个维观察数据，把握总体了。

5.3.2 在 Excel 中浏览该多维数据集

第一步,通过选择“数据”功能卡“自其他来源”的“来自分析服务”打开数据连接向导,选择分析服务中的“A 市烟草公司”多维数据库。

第二步,设置数据透视图的“数值”字段为“销售金额(价税合计)”和“销售税额”,分类字段为“销售日期”,展开 2003 年各个月份,观察总体,发现 12 月份问题最大,如图 5-13 所示。

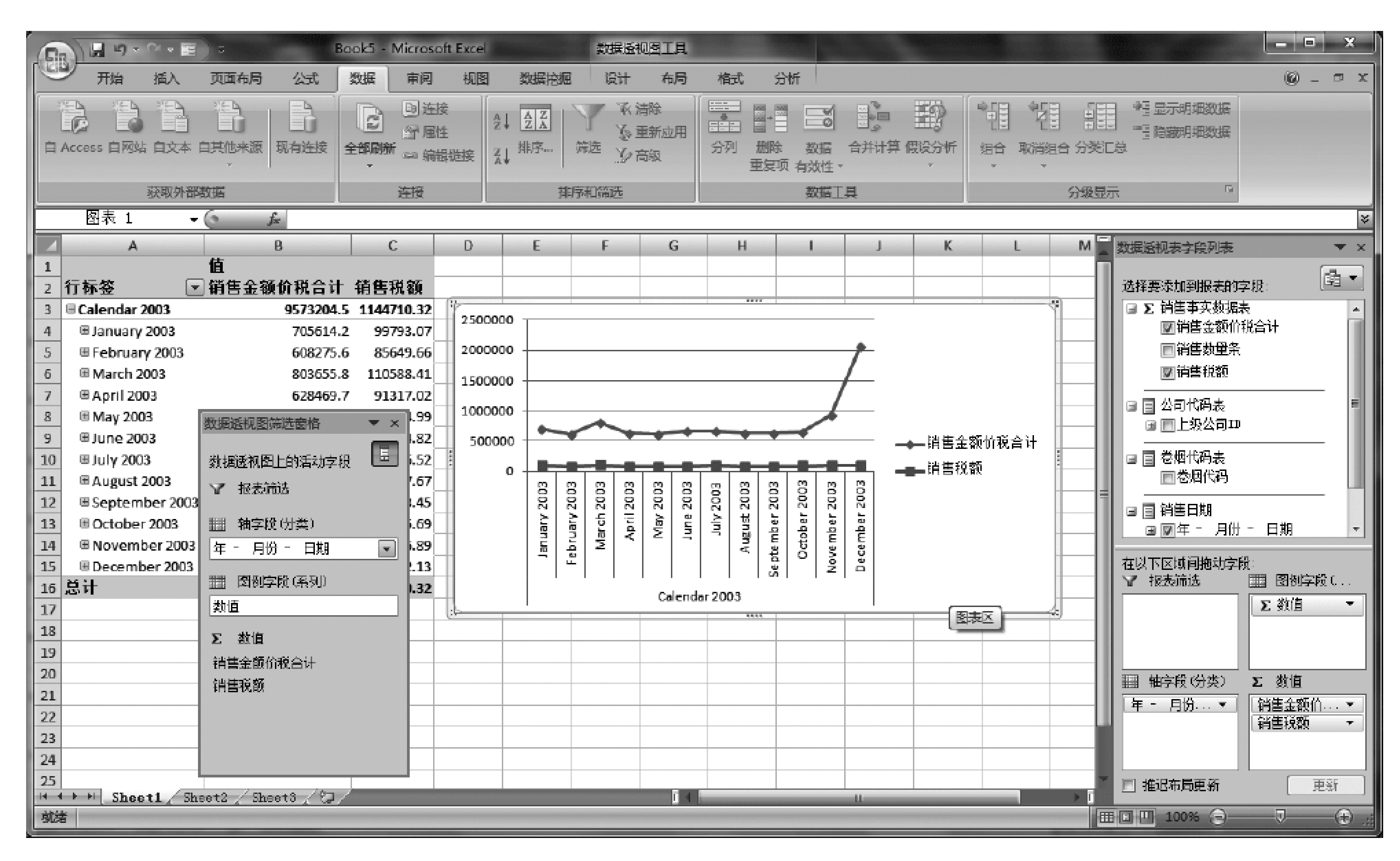


图 5-13 观察总体

第三步,12 月份的问题是由哪个品牌产生的呢? 接着以 12 月份为筛选字段,设置筛选“December 2003”,添加卷烟品牌为分类字段,发现 H532905(黄)的问题最大,如图 5-14 所示。

第四步,查找是哪家销售公司导致的异常? 以 H532905(黄)卷烟品牌作为图例字段,以烟草公司作为分类字段,查找到底哪家烟草公司导致该品牌销售税额和销售金额异常。由于卷烟品牌很多,所以很难从列表中选择到“H532905(黄)”,此时我们希望列标签列表是有序的,这样就方便查找特定成员了。回到“卷烟品牌”维度设计器,把“卷烟代码”属性的 OrderBy 属性值由默认的 Key 修改为 Name。重新处理多维数据集。如果出现错误“处理时找到重复的属性键:表:卷烟品牌,列:卷烟代码,值:340106”,其含义是在卷烟品牌维表中含有重复键值“340106”。打开 Access 数据库,在“卷烟代码”列上设置筛选:值等于“340106”,发现有重复值,此时删除其中一行即可。

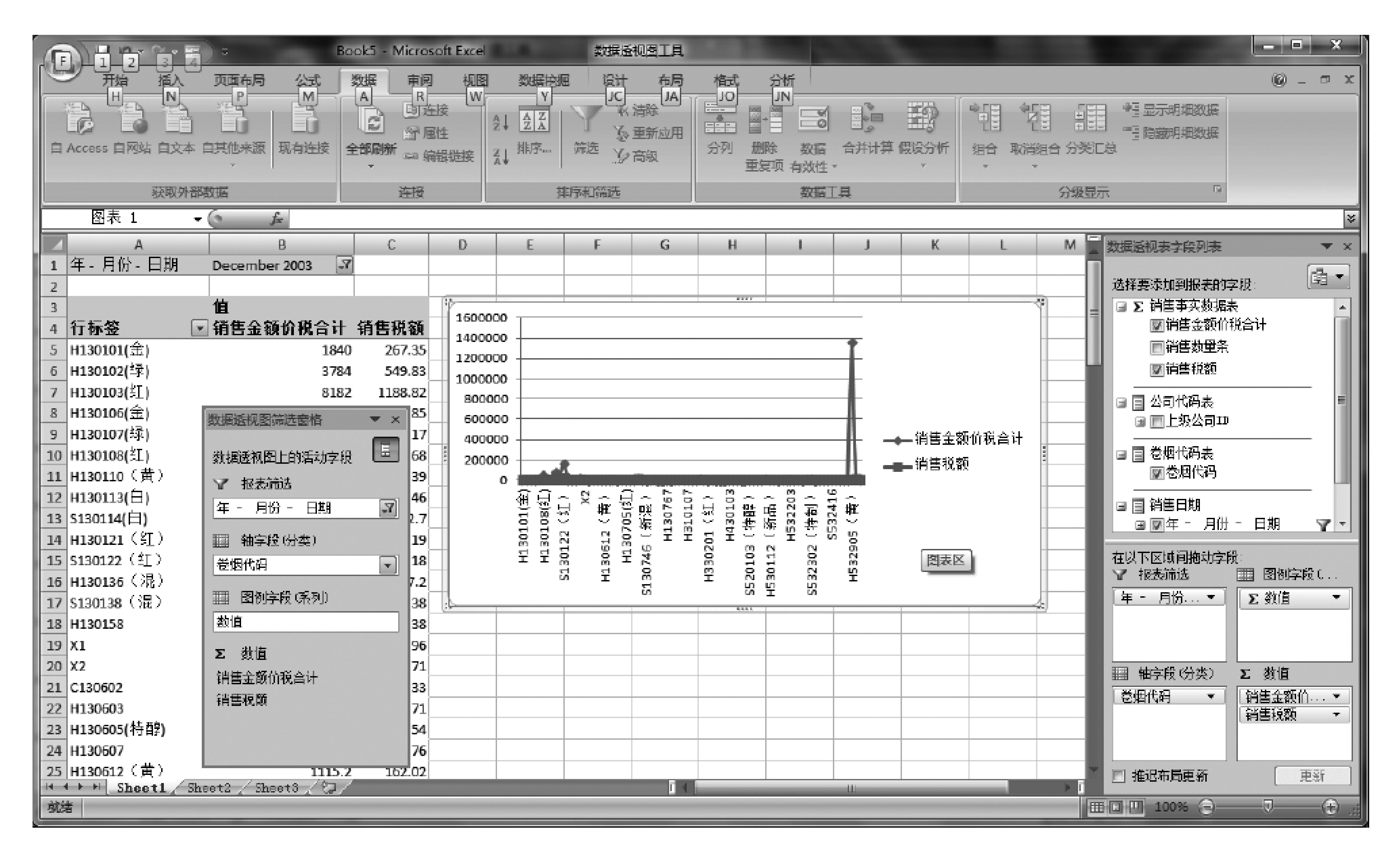


图 5-14　筛选



挖掘型分析

传统审计思路形成的来源主要有：法律法规、被审计单位的业务逻辑、数据间的逻辑关系、被审计数据与外部相关数据的映射和审计经验等。数据挖掘技术的发展和应用为审计思路的形成提供了一条新的途径：让计算机自动地或者半自动地从海量数据中发现潜在的联系或者规律，审计人员再对计算机发现的内容进行确认，进而发现审计线索，这就是数据挖掘在审计实务中应用的基本思路。

6.1 数据挖掘

数据挖掘是从大量的、不完全的、有噪声的、模糊的、随机的数据中抽取出隐含在其中的、人们事先不知道的、但又是潜在有用的信息和知识的过程。

数据挖掘过程(图 6-1)包括 5 个步骤：获取数据、准备数据、建立模型、评估模型和应用模型。

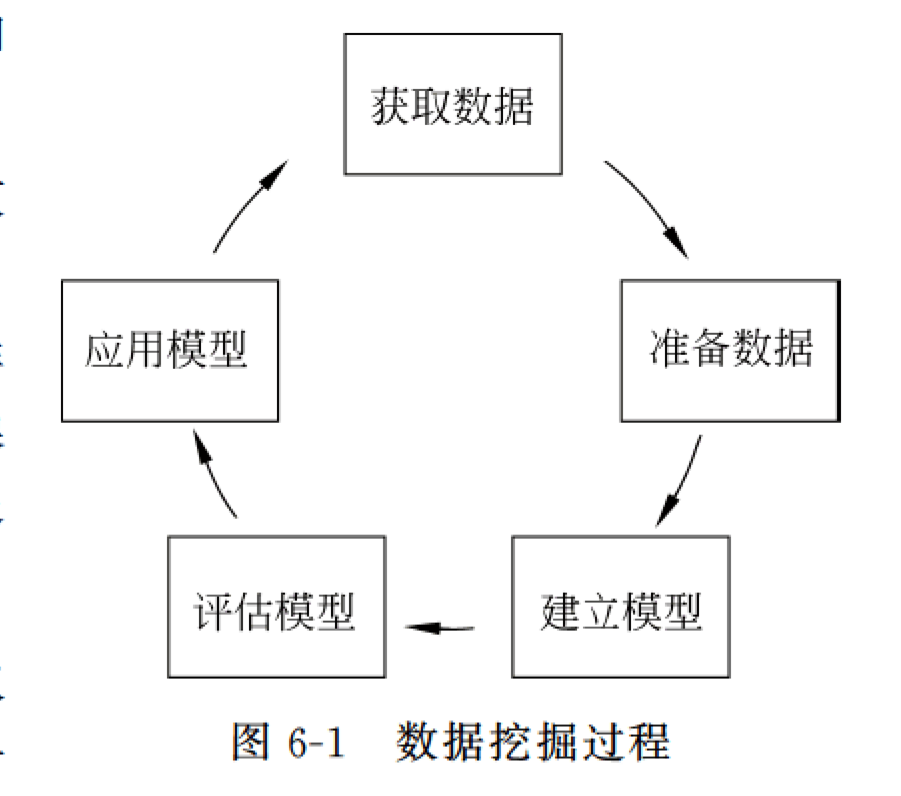


图 6-1 数据挖掘过程

获取数据指的是从被审计单位取得各种各样的数据，包括从各种数据库服务器或者分析服务器中导出的数据、Access 数据库、Excel 文件、各种格式的文本文件、XML 文件等。

准备数据指的是对数据进行清洗、转换、抽取和验证，使之符合数据挖掘模型的要求。清洗指对缺失的、重复的、无效的、不一致的数据进行处理，提高数据质量，并统一量纲。还有一种清洗情形是噪声清除，如“币种”，当大多数事例都是人民币或者美元，大量的币种，如欧元、澳元等没有几个事例。这种情况下，对于数据挖掘来说，具有很少事例支持的属性值对于模式发现没有什么贡献，甚至带来负面影响，所以可以作为噪声数据清除。对具有连续值的属性，则可以通过设置最小值和最大值清除稀疏的事例。转换包括数据类型转换和数据内容转换。不同的数据源使用的数据类型不同，例如，Access 中对于字符数据使用“文本”类型，而 SQL Server 中使用 CHAR 或者 VARCHAR。需要选择合适的映射进行转换。又如，把出生日期转换成为年龄就属于内容转换。同一个实体可能使用不同的表示。也就是说，同样的属性值，可能有不同形式。例如，同样的一所学校“华北电力大学”，可能出现 3 种不同的取值：“华电”“华北电力”

“华北电力大学”，需要把这些值转换成唯一的取值，这也属于内容转换。数据抽取就是选择有关系的表、视图、相关的字段以及相关的行。一般要对数据的概率与统计特征做一般性了解，以决定挖掘需要抽取的行数。验证是指通过检验、确认、清洗、转换后的数据与之前数据的一致性以及数据的完整性。例如，在固定资产表中存在关系：资产净值＋累计折旧＝资产原值，在数据准备阶段需要对这类关系进行验证。经过数据准备，得到的是一个事例表(case table)和多个嵌套表(nested table)。

把全部事例划分成两个子集：训练集和测试集。训练集也称为学习集，是分析服务使用相关的学习算法获取知识的过程。数据挖掘的结果知识是隐含的、事先未知的、潜在有用的信息。知识可表示为概念(concepts)、分类(classification)、规则(rules)、规律(regulations)、模式(patterns)等形式。有的知识可以显式地使用某种知识表示形式表示，如产生式；有的无法用显式的方法进行描述，如用神经网络挖掘出的模式是通过连接权值体现出来的。知识发现的最后一步是结果的表达和解释，该步骤负责将挖掘的模式用更容易理解的方式，如图形、自然语言和可视化技术等展现在用户面前。测试事例集用来评估模型的准确性。

建立模型包括识别数据挖掘方法、选择挖掘算法、设置算法参数、根据挖掘算法选择设置输入属性和设置输出属性、指定测试数据百分比等。训练数据挖掘模型是一个机器学习的过程，这个过程从数据中抽取出了知识，并以决策树、规则等形式表示这些知识。数据挖掘分析模型包括数据挖掘结构和数据挖掘算法。数据挖掘结构中定义了作为输入的事例表；数据挖掘算法从训练集中寻找知识，算法要求定义输入列和预测列。

设置了输入列和输出列，以及数据挖掘算法后，数据挖掘模型就定义好了。建立一个适合业务目标的数据挖掘模型是审计数据挖掘成功的关键，而设置合适的算法参数则影响模型的处理效率和结果的可靠程度。设置算法参数要求对算法本身有较深入的理解。也可以多次尝试，选择最优的参数。

用来进行有监督式数据挖掘的数据通常被随机分成两部分(通常的比例是70％和30％)，一部分作为训练集，另一部分作为测试集。确保训练集和测试集的随机性对于模型的准确率十分重要。可以使用随机抽样或者过度抽样减小数据的规模，以减少模型处理时间的消耗。如果随机抽样策略导致一个测试集正常事例和欺诈事例的分布不同，则应使用分层抽样策略，以确保所有样本都服从数据原始的分布。

抽样(sampling)就是从组成某个总体的所有事例集合中按照一定方式抽出部分事例的过程。简单随机抽样(simple random sampling)是概率抽样的基本形式，它按照等概率原则直接从 N 个事例的总体中随机抽取 n 个组成样本；分层抽样(stratified sampling)将总体样本按其属性特征(如性别、职业、年龄、地域)分成若干类型或层次，然后在各个类型或层次中采用简单随机抽样抽取子样本，最后将子样本合起来组成总体的样本。其特点是，通过划类分层，增大了各类型中单位间的共同性，容易抽出具有代表性的样本。该方法适用于总体情况复杂，各类别之间差异较大，类别较多的情况。分层抽样的优点是，样本代表性好，抽样误差小；过度抽样就是加大样本在总体中的比例。因为在样本比例太小的情况下，如果直接采用某种模型(如决策树、神经网络等)，可能会因为数据概率太小而导致模型失效，这种过度抽样必须谨慎使用。

处理数据挖掘模型是建立模型环节的最后一步。处理的过程是让计算机根据所设置的数据挖掘模型，从数据中发现审计人员感兴趣的知识的过程。根据使用的数据规模的不同以及算法和算法参数不同，处理的时间可长可短，长的可能需要一两天，短的只需要几分钟。

评估模型指通过挖掘准确性图（Mining Accuracy Chart）等工具，使用测试数据评估模型。通常使用准确性、可靠性和有用性对数据挖掘结果进行度量。“准确性”是模型产生的结果与所测试数据中的结果一致性程度的度量。“可靠性”是评估数据挖掘模型在不同测试数据集上测试结果的一致性度量。如果无论提供什么样的测试数据，数据挖掘模型的准确率都差不多，则该数据挖掘模型是可靠的。“有用性”是说明模型产生的结果是否有用的度量。如果经过评估，模型的准确性、可靠性和有用性能够被接受，则可以应用该模型到事例集上进行查询。这个阶段往往需要结合使用 SQL 语句。

应用模型指把模型应用于新的事例集。

数据挖掘过程是一个反复的过程，每一个步骤如果没有达到预期目标，都需要回到前面的步骤，重新调整并执行。

6.2 审计数据挖掘分析

从审计角度看，数据挖掘就是根据事先明确的审计目标，对被审单位的大量业务数据进行分析，揭示其中潜在的逻辑关系和规律，并进而形成明确而有效的审计思路的过程。例如，银行贷款五级分类真实性情况审计。由于种种原因，某些事实上的不良贷款（次级、可疑或损失）被商业银行人为地划归正常贷款分类（正常或关注）中。按照传统的审计思路，需要与被审单位了解贷款五级分类规则，然后根据被审单位提供的规则逐一核实各笔贷款的分类是否正确。挖掘型分析途径是先让计算机从被审单位提供的大量可信贷款数据中分析出贷款五级分类规则，然后使用该规则应用于可疑贷款数据集，从而发现审计线索。

数据挖掘途径的计算机审计真正难点在于定义业务问题，这需要审计人员了解数据挖掘能做什么，从而提出问题。当数据量较小时，即使审计人员事先并不知道数据间的联系和规律，也可以通过简单的查询和分析进行把握，无须数据挖掘。当数据量相当庞大，使用 SQL 查询的方法和多维分析的方法都难以发现数据背后隐藏的规律时，利用数据挖掘就显得非常必要了。

实施数据挖掘的步骤包括定义审计业务问题、数据准备、数据挖掘、结果分析。

定义审计业务问题即明确指出数据挖掘的目标。例如，让计算机在真实贷款事例集上自动总结归纳出贷款五级分类规则。数据准备包括数据采集、数据清洗和数据转换。

可以在关系型数据上进行数据挖掘，也可以在多维数据集上进行数据挖掘。数据挖掘模型设计是根据数据挖掘的目标和数据特征，选择合适的模型。建立一个适合数据挖掘算法的分析模型是数据挖掘成功的关键。关系型挖掘模型包括事例表、数据挖掘算法、事例键表（包含事例键列的表）、事例键列、输入列（依据）、可预测列（结果）等。OLAP 数据挖掘模型包括源多维数据集、数据挖掘方法、将分析的事例维和级别、最初的预测实体、

训练数据等。预测实体是源多维数据集的一个度量值，或是事例级别的一个成员属性，也可以是另一个维度的成员。

6.3　数据挖掘算法

数据挖掘有分类（classification）、聚类（clustering）、关联（associate）、序列模式（sequence pattern）等问题。不同的问题由不同的算法解决。

分类是最基本的数据挖掘方法之一。分类就是把一些事例映射到给定类别中的某一个类别中的过程。实施映射前，一般利用一定的分类算法，从样本中计算得到分类规则，再依据该规则在另一组事例上进行类别的划分。例如，根据信用卡支付事例判断哪些客户具有良好的信用。分类问题的特点是根据事例的某些属性，估计一个特定属性的值。常见的分类算法有朴素贝叶斯（Naïve Bayes）、决策树（Decision Tree）、K最近邻（k-NearestNeighbor，kNN）、神经网络分类（Neural Networks）、支持向量机（Support Vector Machine，SVM）等。通常将所有数据分成训练集和测试集，在训练集上得到模型，在测试集上得到准确率。确保训练集和测试集的随机性对于模型的准确率十分重要。为了避免特殊性，在数据挖掘工具中使用交叉检验的方法。在这种方法中，整个数据集被等分成10个子集，称为小包（folds），每一小包都作为测试集，其余的包作为训练集。将10个试验的平均结果作为最终结果。

聚类就是依据样本特征将事例分到多个簇中，使得同一个簇中的事例"相似"，而与其他簇中的事例"不相似"。一种比较直观的聚类做法是先选取若干事例代表（可能是随机选择），然后计算其他事例与这些代表的距离，把事例划分到距离最近的代表所在的簇中。在审计数据挖掘中，不属于任何簇、或者事例较少的簇，往往就是审计疑点。常见的聚类算法有K均值（K-means）、期望最大化（EM）、自组织图（SOM）、层次聚类（hierarchical clustering）、相异矩阵计算（dissimilarity matrix calculation）、模糊C-均值（fuzzy C-means）、围绕中心点划分（partitioning around medoids）等。

和聚类非常相关的数据挖掘问题是异常检测（outlier detection）。聚类是相似对象的分组，然而，可能存在一些对象不属于任何聚类，这些对象就是异常对象。根据不同的业务领域，这些异常对象极有可能是欺诈。

当评价一个聚类模型时，需要考虑以下几方面内容：各个聚类彼此之间的相似程度；事例属于最相似聚类的程度；是否存在异常记录；某个聚类的典型记录（聚类的特征）；不同聚类之间的差别。

散点图用来分析各个事例到聚类中心的距离。雷达图可以用来评估不同聚类间的距离。

关联分析是从大量事务数据中发现项与项之间的关联，是早期应用的数据挖掘方法之一。关联规则模型很容易理解，也容易实施。一个事务中包含了若干项（item）。关联规则指这些项之间隐含的相互关系。关联分析的任务是发现项间的关联规则。Apriori是一种最有影响的挖掘布尔关联规则频繁项集的算法，是一种基于两阶段频集思想的递推算法；FP-Tree是一种不产生候选挖掘频繁项集的算法。

对于不同业务领域,可通过关联工具解决的业务问题有:什么能量类型会一起产生;病人药物和治疗效果之间存在的关系;哪些基因序列通常一起出现;哪些航班经常一起延误;哪些汽车零件经常一起发生故障等。

关联规则模型很容易理解,但是往往会产生大量无价值的规则。

序列模式挖掘就是要找出序列中所有的序列模式。将重点放在发现项之间的一定的因果关系。给定一个由不同序列组成的集合,其中每个序列由不同的元素按顺序排列,每个元素由不同项目组成,同时指定一个最小支持度阈值。序列模式挖掘就是找出所有的频繁子序列,即该子序列在序列集中的出现频率不低于最小支持度阈值。序列模式挖掘的主要算法有 GSP、PrefixSpan 等。GSP(Generalized Sequential Patterns)算法类似于 Apriori 算法,PrefixSpan(Prefix-project Sequential Pattern mining)算法采用分治的思想,不断产生序列数据库的多个更小的投影数据库,然后在各个投影数据库上进行序列模式挖掘。序列模式挖掘是数据挖掘一个重要的研究领域。例如,通过对 Web 日志挖掘可以发现访问规律等。

不同的数据挖掘方法能够解决不同类型的问题。表 6-1 是部分数据挖掘算法的一些典型优点和缺点。数据挖掘是非常活跃的研究领域,表中的评判也可能会随着研究的进展而更新。

表 6-1 数据挖掘算法的一些典型优点和缺点

算　法	效　果	可解释性	可操作性	结果产生时间	结果可视化
决策树	好	非常好	好	快	是
聚类	好	非常好	好	快	是
关联规则	好	非常好	好	慢	是
神经网络	非常好	不好	不好	慢	否
Logistic 回归	好	非常好	非常好	快	否

不同的算法对数据有不同的结构要求,审计人员实践中往往需要根据要解决的具体问题选择合适的方法。例如,如果是分类问题,就选用分类方法;如果是聚类问题,就选用聚类方法。方法确定后,再根据事例属性的类型选用合适的算法。例如,在回归方法中,如果输出属性是二值的,则适合选用 Logistic 回归。如果不确定使用哪个算法,可以对工具中的候选算法进行尝试,选择挖掘结果最好的算法。

例如,根据审计工作方案要求,审计组要对 S 省商业银行贷款 5 级分类的真实性进行核实。主要问题是商业银行把应归于不良的贷款划归正常贷款。

贷款 5 级分类即正常、关注、次级、可疑和损失。前两级属于正常贷款,后 3 级属于不良贷款。由于种种原因,某些事实上的不良贷款被商业银行划归为正常贷款的分类中,从而掩盖了信贷风险。

按照通常的审计思路,审计人员需要解决两个问题:一是通过调查,学习该省银行的 5 级分类规则;二是把学习到的 5 级分类规则应用到全省每一笔贷款事例中,核实事例是否真实。实际上,这两个问题都很难解决,而且不现实。

一个新的思路是，让计算机自动从大量贷款事例中学习到 5 级分类规则，然后应用该规则核对可疑贷款事例。根据调查得知，5 级分类依据主要包含两个方面：一是法人有关属性，如经营状况、管理特征；二是贷款本身有关属性，如担保方式等。学习 5 级分类规则如图 6-2 所示。

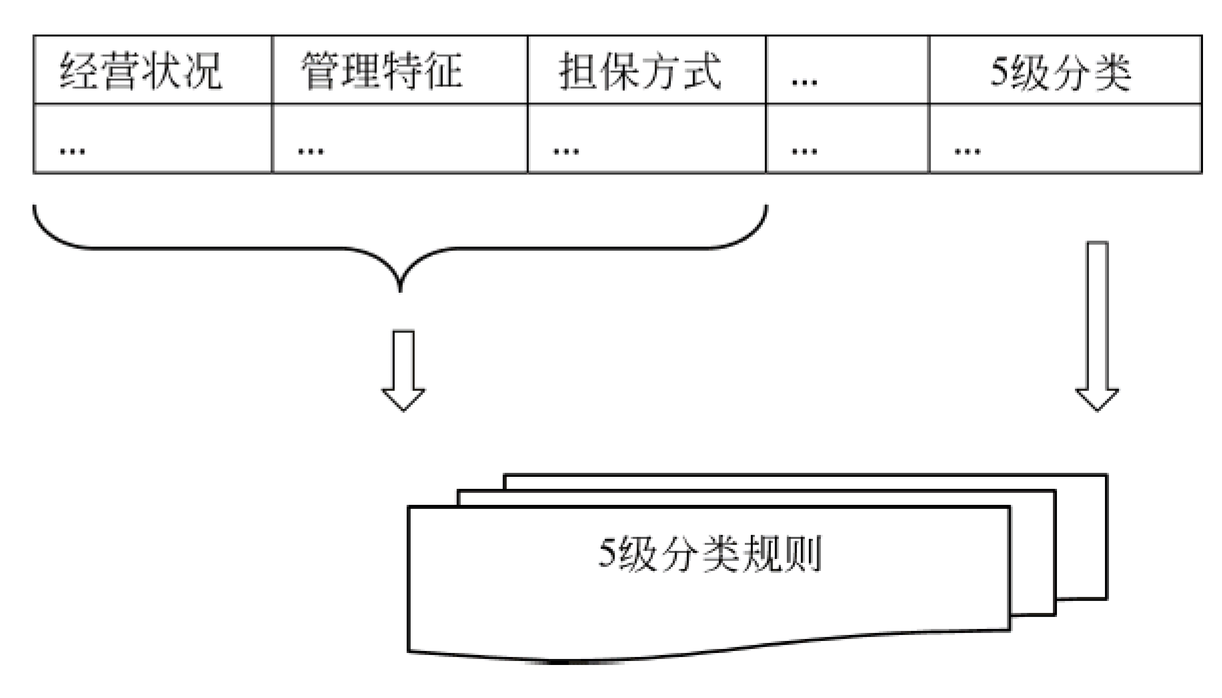


图 6-2 学习 5 级分类规则

根据审计目标（对 S 省商业银行 5 级分类真实性进行核实）和数据挖掘途径的计算机审计的一般过程模型，审计人员现在能够提出明确的业务问题：通过数据挖掘从大量贷款事例中发现分类规则，然后确定这是一个分类数据挖掘任务。对于分类数据挖掘任务的过程是：首先把从被审单位采集到的全省约 240 914 万个贷款事例分成两个集合，一个是可信事例集合，另一个是可疑事例集合。审计人员在可信事例集合上让计算机自动学习 5 级分类规则（当然要设置从哪些列中学习），再把此规则应用于可疑事例集进行重新分类，然后比较新分类和原分类有无不同。两个分类不同的贷款事例就成为审计重点。

下面选择支持分类的算法。分析服务中提供了 Microsoft 决策树算法。

给定一个数据集，让分析服务根据决策树算法发现 5 级分类规则是首要的工作，该工作称为训练，用来进行训练的数据集称为训练集。将发现的规则应用于另外一个数据集从而发现问题，该数据集称为预测集。

训练集必须真实可信。可把审计年度发放且已经到期的、当前归属正常和关注的贷款事例作为预测集（可疑事例），其余事例作为训练集（可信事例）。根据从可信事例集中学习到的 5 级分类规则对可疑事例集重新分类，重新分类后的类标签与原来的类标签不一致的贷款事例就称为疑点。

关联规则算法检测具有类似购物事务中的隐藏的物品间的规则。简单地说，一条规则就是一个“如果…那么…”语句。出现在一个事务中的物品称为“项（item）”，一个事务中的若干个物品称为“项集（itemset）”。

在关联规则中，支持度和可信度能够比较直接地形容关联规则的性质。任意给出事例中的两个项集，它们之间都存在关联规则，只不过程度有所不同。如果不考虑关联规则的支持度和可信度，那么在事务数据库中可以发现无穷多的关联规则。事实上，人们一般只对满足一定的支持度和可信度的关联规则感兴趣。因此，为了发现有意义的关联规则，需要给定两个阈值：最小支持度和最小可信度，前者规定了关联规则必须满足的最小支持度；后者规定了关联规则必须满足的最小可信度。一般称满足一定要求（如较大的支持

度和可信度)的规则为强规则(strong rules)。

在关联规则的挖掘中要做好数据准备工作。数据准备将直接影响到问题的复杂度及目标的实现。另外,注意选取恰当的最小支持度和最小可信度。这依赖于用户对目标的估计,如果取值过小,会发现大量无用的规则;如果取值过大,又有可能找不到规则。

数据挖掘工具能够发现满足条件的关联规则,但它不能判定关联规则的实际意义。对关联规则的理解,仍然需要熟悉业务背景。在发现的关联规则中,可能有两个主观上认为没有多大关系的项,它们的关联规则支持度和可信度却很高,需要根据业务知识、经验,从各个角度判断这是一个偶然现象或有其内在的合理性;反之,可能有主观上认为关系密切的项,结果却显示它们之间关联不强。

例如,审计人员可以通过挖掘医保报销数据中的用药规则,检测出不符合规则的“假住院”审计疑点。

在 Microsoft SQL Server 企业管理器中建立查询,选择医疗机构 H000 和 H001 报销事例进行学习。使用报销单据主表和入院登记表建立视图 VEXAMPLE。SQL 语句如下。

```
CREATE VIEW VEXAMPLES
AS
SELECT 医疗机构码,住院序号,住院诊断码,诊断码,单据号,单据发生日期,金额
FROM 报销单据主表 as B JOIN 入院登记 AS R ON B.住院序号 =R.住院序号
WHERE R.医疗机构代码 IN ('H000','H001') AND R.诊断码 is not null
```

准备冠心病住院事例以及消费明细:

```
CREATE VIEW VEXAMPLESGXBH000H001
AS
SELECT 医疗机构码 +住院序号 AS 住院号,单据号
FROM VEXAMPLES
WHERE 住院诊断码 ='6703'
```

其中,“6703”是冠心病的诊断码。

从报销单明细表中筛选这两家医疗机构的住院报销明细,发现这两家医院的冠心病报销有 1 173 355 条记录。

```
CREATE VIEW VDETAILS
AS
SELECT 医疗机构代码,医疗机构代码 +单据号 AS 单据号,项目名称,单价,数量
FROM dbo.报销单明细表
WHERE 医疗机构代码 IN ('H000','H001')
```

以冠心病住院为事例,通过报销单据把住院事例和消费明细进行连接,创建用于关联规则分析的视图。

```
CREATE VIEW VASSGXBH000H001ZY
AS
```

```
SELECT M.住院号, D.项目名称 AS 项目
FROM VEXAMPLESGXBH000H001 AS M JOIN VDETAILS AS D ON M.单据号 = D.单据号
```

至此，用于冠心病消费项目关联规则分析的数据结构和数据已经准备完毕。

启动 SQL Server 数据库服务和分析服务，在 Excel 中选择“数据”功能区，单击“获取外部数据”，然后单击“自其他来源”“来自 SQL Server”。在接下来打开的“数据连接向导”中输入数据库服务器的名字，保持默认的登录凭据不变。

接下来按照向导指示，从数据库中选择“VASSGXBH000H001ZY”视图作为数据源导入到 Excel sheet1 中并作为一个表。把 sheet1 改名为“H000H001 冠心病”。

Excel 的关联规则要求数据必须按照事务 ID 排序。所以，在“数据”功能区，单击“排序”，设置事务 ID 按照升序排序。

然后从 Excel 中选择“数据挖掘”功能区，单击“关联”，弹出关联规则数据挖掘向导。一般保持默认选项：选择“H000H001 冠心病”工作单中的“VASSGXBH000H001ZY”表作为数据源。

在接下来的设置中，Excel 已经聪明地识别出“住院号”是关联规则算法中的事务标识(ID)，项目是关联规则算法中的“项”。如图 6-3 所示，在“阈值”设置中，“最低支持等于 10”的意思是包含某个项集的事务至少有 10 个；“最小规则概率＝40％”的意思是某个关联规则的可能性。例如，如果冠心病病人吃 A 药，那么他也吃 B 药的可能性是 40％。

保持默认设置不变，单击“下一步”按钮，如图 6-3 所示。

关联 向导
关联
选择跨多行标识事务的列和包含希望查找关联性的项的列。数据应按事务 ID 排序。
事务 ID(T)： 住院号
项(I)： 项目
阈值
最低支持(M)： 10 百分比(E) 项(S)
最小规则概率(R)： 40.0 %
参数(P)... ＜上一步(B) 下一步(N)＞ 取消

图 6-3 关联

把“模型名称”改为“冠心病消费项目关联分析”，单击“完成”按钮。Excel 开始调用分析服务的关联规则算法对 Excel 表中的数据进行学习。这个过程可能会持续较长时间。

Excel 学习结束后，自动弹出“浏览”对话框，显示学习的结果。其中，“规则”选项卡列出了学习到的所有规则，如图 6-4 所示。例如，规则“泵->导管置入套件”的意思是，如

果治疗中使用了“泵”，那么也会使用“导管置入套件”。该规则的概率是 0.893，重要性是 1.637。如果重要性为 0，则表示两个项目毫无关联。重要性就是在不使用“导管置入套件”情况下使用“泵”的概率去除该规则的概率。重要性的值越大，说明当使用“泵”时，使用“导管置入套件”的可信度越高。

可以根据需要设置筛选满足“最小概率”和“最低重要性”的规则。图 6-4 显示了概率大于或等于 0.8，重要性大于或等于 0.6 的所有规则。



图 6-4　浏览规则

“项集”选项卡显示了满足最低支持和最小项集大小的项集列表，如图 6-5 所示。例如，“住院诊查费”项集的大小是 1，即该项集中只有一个项目；项集的支持度是 374，意思是有 374 个住院事例中有该消费项目。

“依赖关系网络”展示了消费项目之间的依赖关系。通过上下拖动左侧的滚动条，可以调节依赖的弱强，从而增加或者减少途中箭头的数目，便于观察。单击某个项目，可以通过颜色显示与其相关的项目。

回到“规则”选项卡，单击左下角的“复制到 Excel”按钮，把所有满足条件的规则复制到 Excel 中。在“规则”工作单上单击“规则”上面的筛选按钮，设置筛选“包含‘泵’”的规则，结果如图 6-6 所示。

把医疗机构 H002 作为预测集，首先按照类似的做法创建视图 VASSGXBH002，整理出 H002 的冠心病所有住院事例(326 个冠心病)和消费项目，应用如下 SQL 语句，查询无“住院诊查费”的住院事例，找到 19 个。

```
SELECT distinct 住院序号
FROM VASSGXBH002
except
```

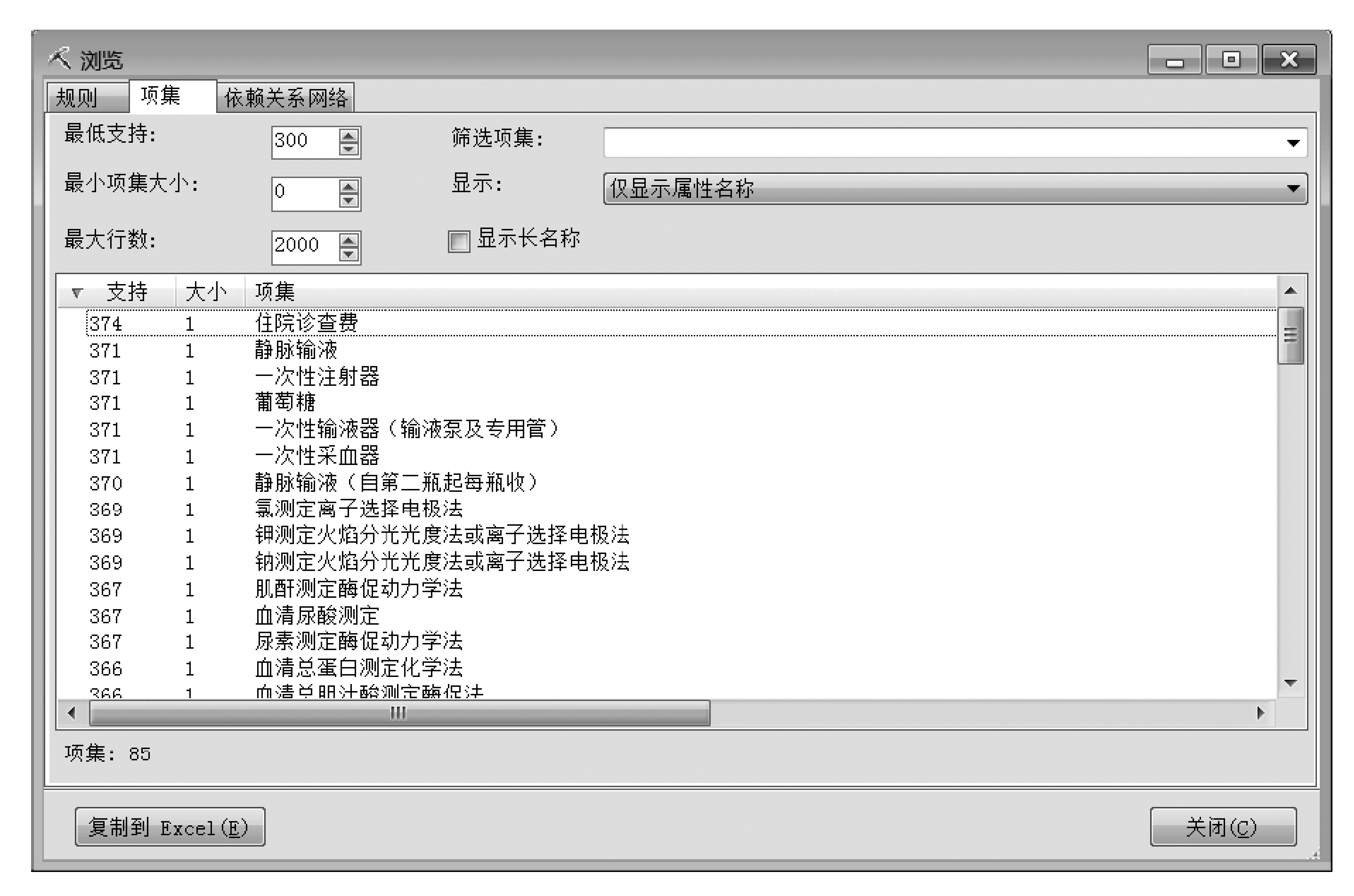


图 6-5　项集


冠心病住院消费项目关联

规则

概率	重要性	规则
81 %	1.83	导管置入套件 -> 泵
89 %	1.64	泵 -> 导管置入套件
82 %	1.17	泵 -> 冠状动脉造影术二次检查及复查
86 %	1.05	泵 -> 动脉穿刺置管术（测压加收）
89 %	1.03	泵 -> 外固定材料
93 %	0.99	泵 -> 支架（介入）
100 %	0.96	泵 -> 导管
82 %	0.86	鼻饲泵囊 -> 鼻饲管置管（注食，注药加收）
93 %	0.77	泵 -> C型臂术中透视
89 %	0.74	泵 -> 动脉穿刺置管术
86 %	0.70	泵 -> 持续有创性血压监测

图 6-6　筛选规则

```
SELECT  DISTINCT 住院序号
FROM VASSGXBH002
WHERE 项目 ='住院诊查费'
```

然后，在这 19 个住院事例中应用关联规则。例如，从无住院诊查费的事例 XXXXXXXX（H002 的住院号）的消费项目中发现其使用了“泵”，根据关联规则：泵->支架（介入）、泵->导管等，发现该住院事例不满足条件，因此审计人员将此次住院列为疑点。

第7章

大数据分析

7.1 大 数 据

当数据集的规模超出传统数据库软件的管理和分析能力，通常达到几十 TB，甚至几个 PB 规模时，被称为“大数据”。一般来说，超大规模数据是指千兆字节 GB（1GB＝1024MB）级的数据；海量数据是指万亿字节或者太字节 TB（1TB＝1024GB）级的数据；大数据则是指千万亿字节或拍字节 PB（1PB＝1024TB）级及其以上（EB、ZB 和 YB）的数据。

大数据有 3 个基本特征：体量（volume）大、流动快（velocity）和多样性（variety）。随着物联网技术、互联网技术不断深入到人类社会的各个角落，人类的社会活动，无论是步行过马路，还是驾车过道口；无论使用社交软件聊天，还是网上浏览购物；无论使用网约车出行，还是到外地住宿餐饮，都被以各种各样形式记录在某个存储器中，从而累积成大数据。

数据可分为 3 大类：文本、音频和视频。从结构角度看，对数据的组织有结构化、半结构化和无结构 3 大类。关系数据库就是典型的以结构化方式管理数据。Excel 也是以结构化方式管理数据。从数据项之间的前驱、后继关系划分，数据结构可分为线性表、树和图。网页（Web page）是典型的半结构化数据，这种形式允许数据模式（schema）动态变化。超文本标记语言（eXtensive Markup Language，XML）描述的数据也是半结构化的。一个文本文件（.txt 文件）仅是字符的集合，一般认为是无结构的。

大数据不仅带来了数据资产，更重要的是改变了思维方式。

首先的变革是大数据允许直接在总体上进行计算，而不必采样。当总体的数据不可得，或者获得的代价无法承受时，只有使用采样技术，使用样本的统计量去估计总体的参数。大数据不仅在数据采集上得到了总体数据，而且在存储和计算上都提供了解决方案。但是，这并不意味着采样和估计失去了价值。某高校曾应用学生食堂就餐的大数据遴选贫困生，但是结果并不准确。所以，大数据只是提供了解决问题的一种视角，但不是唯一的。

其次，关注相关，不必关注因果。人类认识世界，首先想知道“是什么”，在此基础上接着想知道“为什么”，这就是追求因果：什么原因导致什么结果。人们要么从结果中追溯原因，要么从原因演绎结果。电视剧《康熙王朝》中有一个情节：康熙小时候得了天花，苏麻喇姑采来“茇茇草”，结果康熙服用 7 天后病愈了。那么“茇茇草”和“治疗天花”之间有

因果关系吗？恐怕这个问题很难回答；但是如果问，“茇茇草”和“治疗天花”之间相关吗？那么回答起来似乎容易一些。大数据使得人们能够意识到许多以前并没有意识到的联系的存在。这里的相关关系不完全是线性相关，应该是一种联系。究竟以什么样的函数表达这种联系，大数据的应用并不关心。一旦完成了大数据相关关系分析，而又不满足于仅仅知道“是什么”，自然会继续研究“因果关系”。因果关系是一种特殊的相关关系。

还有，大数据基础上的简单算法比小数据基础上的复杂算法更加有效。例如，谷歌使用了不同语言翻译的质量参差不齐的数十亿对译文档作为语料库训练其翻译系统，这些文档包括企业官方的不同语言版本的主页以及国际组织发布的官方报告的不同语言版本，不进行过滤，保留不完整的句子、语法错误、拼写错误等，也不用人工进行注解。庞大而复杂的语料反而使得谷歌翻译系统的质量得到了提升。

7.2 大数据审计分析

大数据为审计提供了全覆盖的机遇，也提出了新的挑战：如何从大体量的、纷繁复杂的经济活动中发现审计疑点和审计线索。例如，如何从海量贷款示例中查找出异常贷款？

“自顶向下、逐层细化”是基于大数据审计分析的一种策略。也就是先把握总体；然后发现疑点，即找到突破口；最后定位事项，分散核实，归结证据。

把握整体就是借助可视化的数据分析工具选取关键的业务、财务特征，以二维或多维形式，对被审计数据进行多维度可视化分析，观测被审计数据在时间、空间上的分布以及变化趋势；比较同比、环比的变化，发现大的波动，确定主题，在整体和宏观层面上通过观察发现规律，寻找异常。

把不同部门的不同主题的数据关联起来、使用适当的特征，利用数据挖掘技术发现疑点。

根据疑点逐一分散调查取证，进行核实。

例如，某审计机关在农村危房改造专项资金审计中首先采集住建、财政、房产、民政、车管、扶贫、新农合等单位的数据，从“危改户收入是否超标”这个角度出发，对建档立卡、工商登记、财政供养、车辆、房产、公积金等多部门数据进行关联分析，从危改户个人维度、家庭成员维度等多个维度观察数据，建立宏观认识。

然后围绕危房改造专项资金的供应和管理，以资金拨付、资金使用效益、危改政策落实以及危改工程推进等为基础，对采集到的多部门数据进行跨年度、跨领域关联比对分析，从资金领取年度、家庭成员、人员类别、人员身份等多维度进行数据挖掘，发现异常事例，通过钻取等操作快速锁定审计疑点。

最后是入户调查，扎实排查疑点。对于锁定的审计疑点进行有针对性的入户调查，重点审查重复享受危改补助、克扣截留补助、以收取其他费用名义索要回扣等问题。全面排查疑点对象的个人申报资料、危改资金银行账户、村委证明、公示材料、危房鉴定书、验收档案等，并深入了解五保户、低保户、残疾人及其他贫困人员享受危改政策情况，归结证据。

基于大数据计算机审计的过程包括：采集、信息抽取和清理、数据集成、建模和分析，以及解释和部署。在不影响对审计线索发现的条件下，往往需要按照一定比例筛选和压

缩原始数据。应妥善定义筛选条件，使得不会抛弃有用的数据；收集到的数据格式往往不满足实际需要，应按照一定的抽取流程把所需的信息从底层数据源抽取出来，然后再以适于分析的结构化形式呈现。有效的大规模分析往往需要从多个数据源收集异质数据。例如，在整合和分析医疗健康记录的同时，可以利用通过互联网获取的环境数据。数据转换和集成工具集帮助审计师解决数据结构和语义上的异质性问题。大数据分析方法与传统的小样本统计分析方法存在根本区别。噪声、动态、异质性、相关性和不可靠比小样本更有价值，综合统计结果通常不仅能克服个体数据的抖动和偏差，还能揭示更可靠的隐藏的知识。审计师必须认真审视在多个分析阶段中做出的各种假设，对大数据分析结果进行合理的解释。

7.3 大数据可视化

人们的决策可分为两种：理性的(rational)和非理性(irrational)的。所谓“理性”，指符合某种逻辑推演；所谓“非理性”，指无法用一种逻辑进行解释。例如，雾霾天气，许多人出门都戴上了 PM2.5 口罩，这就是一种理性的决策；又如，在青年偶像剧中经常会听到这样的对白：“我就是喜欢他(她)”。这就是一种非理性的决策。人类的决策很多情况下是靠“直觉”完成的，这种“直觉”也是非理性的，但是，直觉往往是正确的。

直觉从哪里来？从人的感知来。在视觉、听觉、触觉、嗅觉、味觉等感知途径中，视觉是最主要的，通过视觉接收的信息不仅多，而且快。为什么用人单位招聘一定要有“面试”环节呢？这就是期望通过视觉捕捉更多的信息，以便决策。

大数据的体量大使得我们无法在“明细”的级别上逐条阅读和审核数据，我们希望在不同“粒度”(granularity)或者“尺度”(scale)上观察数据；大数据的流量快也使得我们无法及时捕捉数据中的特征、关联、变化模式。可视化是把信息传递给人类的有效途径之一。可视化指使用不同形状、大小、颜色、位置、方向、姿态、亮度、灰度、背景、场景等视觉元素表达数据中隐含的语义。

图 7-1 展示了一个可视化例子。数据来自“河北人才网”的招聘求职信息。把某个时间区间的招聘信息中关于岗位要求的文字描述全部提取出来，分词后统计词频，并使用 R 的词云包制作。

该图使用了颜色、字号和位置展现不同词汇的频率。词频越高，字号越大，位置越居中。从图 7-1 中可以迅速获取信息：企业招聘最关心两件事，一是经验；二是能力。

图 7-2 展示了计算机专业求职者与数学专业求职者的数量随年龄的变化情况。很快能够发现，两个专业的求职者年龄分布不同，计算机类在 29 岁出现求职高峰，并且计算机类在 35 岁还出现小高峰；而数学类求职者在 27 岁。

研究生毕业一般在 27 岁。求职高峰与学历相关吗？有因果吗？这是值得进一步探究的问题。

一般来说，人们在看到一个折线图、饼图或者条形图时，容易迅速发现变化趋势或者特征。这对于决策特别重要，但要根据数据的时间、空间等特点设计合适的可视化形式，如统计图(chart)、图(diagram)、图形(graph)、地图(map)、动画(animation)等。数据可视

图 7-1　词云

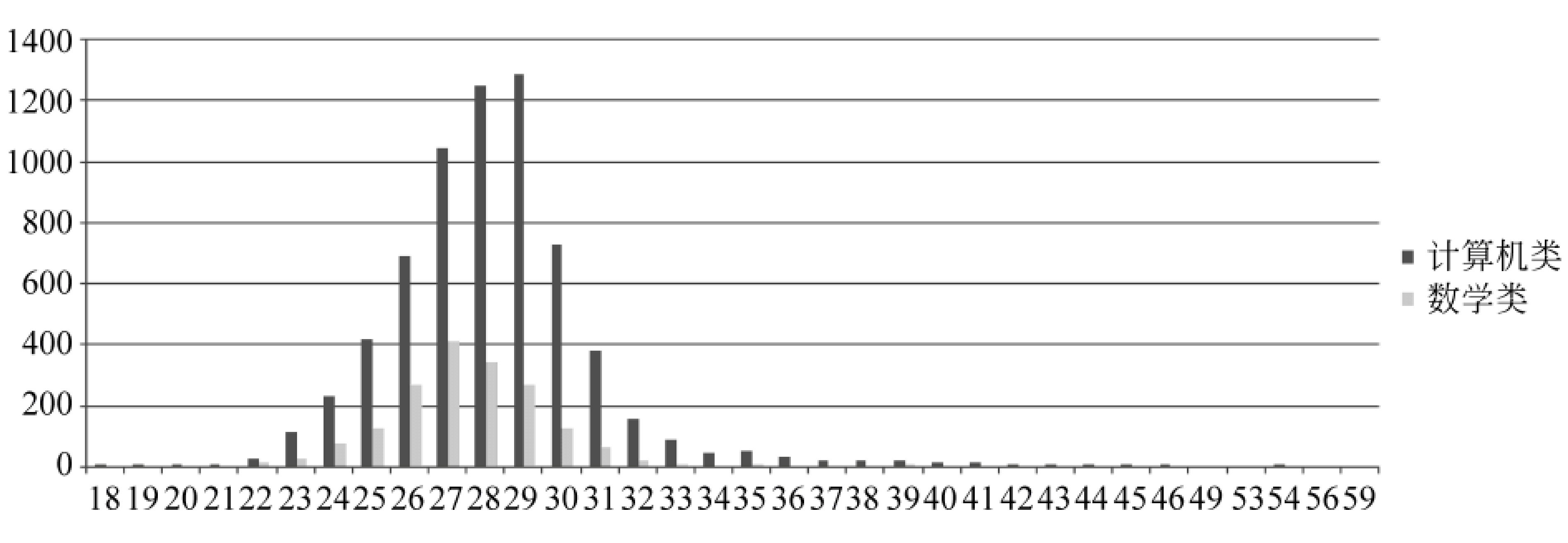


图 7-2　计算机类与数学类求职者数量随年龄变化情况

化的目的是帮助人们理解数据，对数据有全面的或者本质的认知。

Excel 透视图在数据集上创建二维或者三维图形。通过观察这些图形，从而发现那些隐藏的、事先未知的和可能有用的信息或模式。散点图、折线图、柱形图和饼图可以实现从多维角度观察数据；树形图从层次角度观察数据。

柱形图用来分析连续量在离散量上的比较，如税额在税务机关上的比较。离散量（如税务机关）安排在横坐标轴上，连续量（如税额）的汇总安排在纵坐标轴上。

柱形图共有 7 种子图表类型：簇状柱形图、堆积柱形图、百分比堆积柱形图、三维簇状柱形图、三维堆积柱形图、三维百分比堆积柱形图和三维柱形图。其中，簇状柱形图是柱形图的基本类型。

折线图用来显示一系列数据在离散字段和时间上的变化趋势。这些趋势包括：递增、递减，增减的速率，增减的规律(如周期性、螺旋性等)，峰值等。折线图也可用来分析多组数据随时间变化的相互作用和相互影响。在折线图中，一般水平轴(X 轴)用来表示时间的推移，并且间隔相同；而垂直轴(Y 轴)代表不同时刻的数据的大小。折线图共有 7 个子图表类型：折线图、堆积折线图、百分比堆积折线图、数据点折线图、堆积数据点折线图、百分比堆积数据点折线图和三维折线图。

例如，通过折线图比较各税务机关在 2002 年、2013 年、2004 年延期纳税批准额的趋势。延期纳税批准额初始状态如图 7-3 所示。

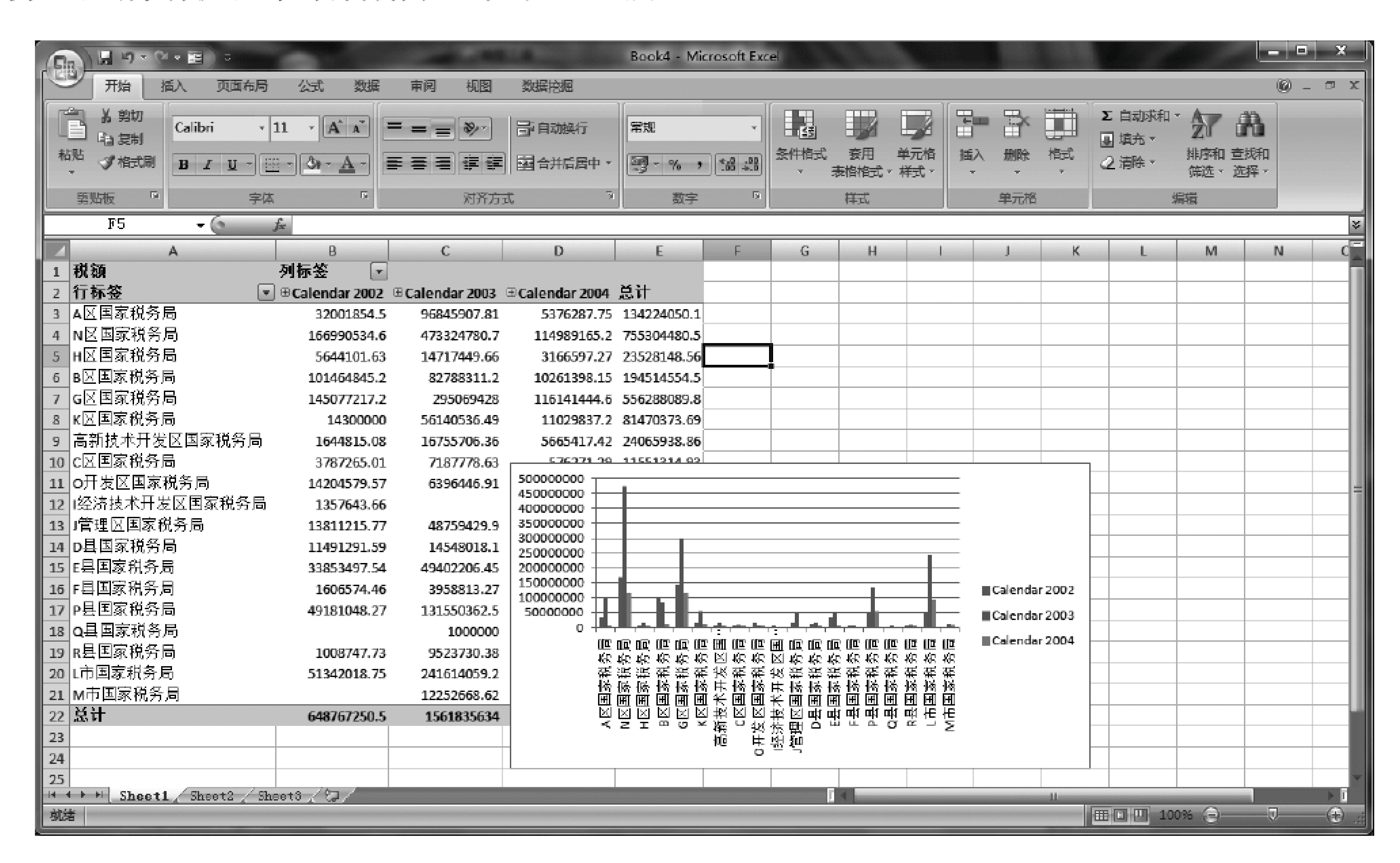


图 7-3　延期纳税批准额初始状态

右击数据透视图区域，选择“更改图表类型”，在“图标类型”对话框中选择“折线图”，单击“确定”按钮，结果如图 7-4 所示。

观察折线的形状可以发现，N 区国家税务局批准延期纳税额的力度在 2013 年变化明显；G 区、L 区和 N 区的延期纳税批准额在 3 年间均高于其他税务机关。

饼图通常用来描述比例、构成等信息。

饼图共有 6 个子图表类型：饼图、三维饼图、复合饼图、分离型饼图、分离型三维饼图和复合条饼图。其中，复合饼图和复合条饼图是在主饼图的一侧生成一个较小的饼图或堆积条形图，用来将其中一个较小的扇形中的比例数据放大表示。

例如，通过饼图观察各税务机关对 2013 年总延期纳税额的贡献程度。

首先在 Excel 数据透视表的“列标签”中设置筛选：2013 年。然后在数据透视图上右击，选择“更改系列图表类型”，在“图标类型”对话框中选择“饼图”，单击“确定”按钮，出现图 7-5。

很快看到，N 区国家税务局贡献最大。把鼠标指针停留在 N 区国家税务局上，出现

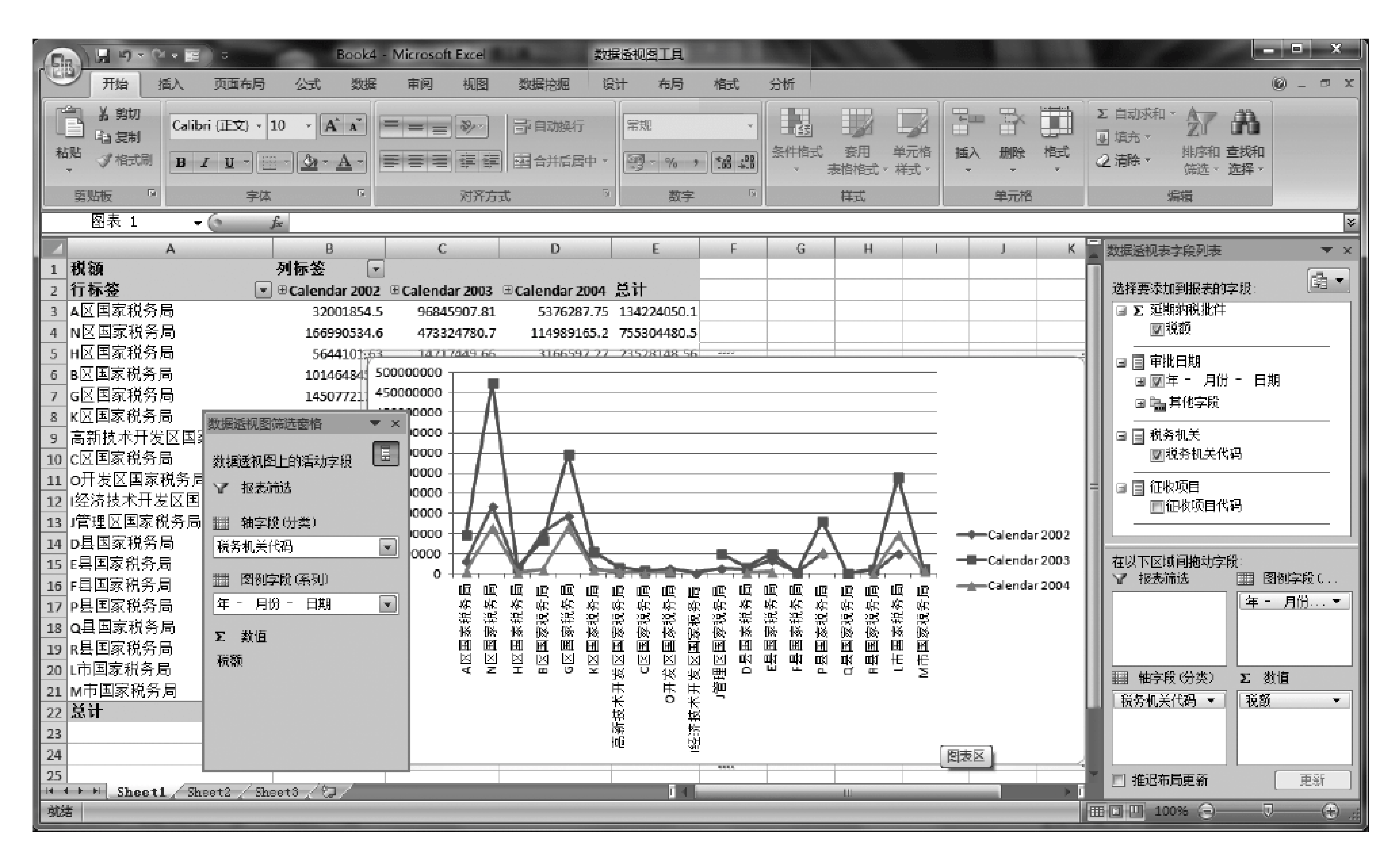


图 7-4　折线图

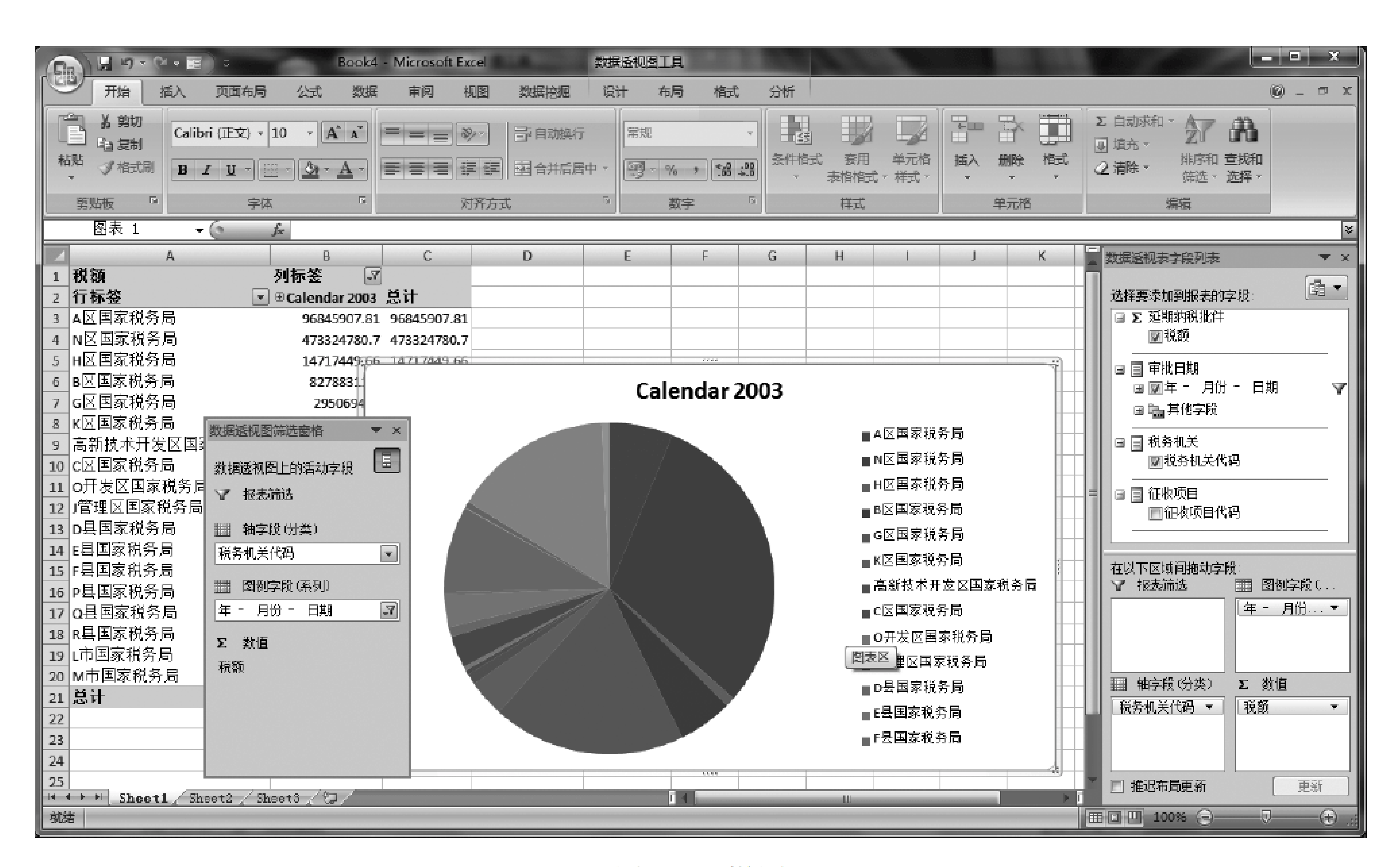


图 7-5　饼图

黄色的提示信息：系列“2013”引用“N 区国家税务局”数值：473324780.7(30%)。这里的 30%值占总数的 30%。

当遇到以下情形时，尤其需要数据可视化：数据的底层结构相同，并可以归类；数据的领域性太强，跨领域理解的难度大；数据的时间和空间跨度大，等等。

图7-6展示了2018年2月11日13时38分22秒时石家庄市的交通状态，该图片使用百度地图获得。



图7-6 城市交通状态

从图7-6中可以看到，拥堵路段出现在二环路上，主要有3处：西二环北段、东二环中段和东北二环。南二环通往石家庄站也有局部拥堵。注意，2018年2月11日是农历腊月廿六，距离中国传统的重要节日——春节只有一周的时间。地图数据和交通数据都是专业性很强的数据，但机动车驾驶人不必关心数据的底层细节，能读懂交通状态即可。

大卫·麦克坎德莱斯说，“可视化是压缩知识的一种方式。”所以，可视化的结果是有帮助的，但是，可视化的过程并不轻松。可视化需要的技术包括：数据的清洗和变换、文本标记、集成多个数据源的数据、集成数据挖掘、集成知识表示和推理技术以及实现普适性等。

参 考 文 献

[1] 周苏，王文. 大数据可视化[M]. 北京：清华大学出版社，2016.
[2] 数据挖掘技巧编写组. 数据挖掘技巧(审计技巧丛书)[M]. 北京：中国时代经济出版社，2016.
[3] 维克托·迈尔-舍恩伯格，肯尼思·库克耶. 大数据时代[M]. 杭州：浙江人民出版社，2013.
[4] Lander J P. R 语言实用数据分析和可视化技术[M]. 蒋家坤，等译. 北京：机械工业出版社，2015.
[5] 董东，梁艳，王立敏，等. 计算机审计的研究现状和挑战[J]. 河北省科学院学报，2011. 28(4)：70-73.
[6] 董东，马志永，庞飞，等. 应用扩展的 UML 活动图建模审计专家经验[J]. 河北师范大学学报，2008，32(5)：599-602.
[7] 陈伟，QIU Robin，刘思峰. 数据库技术在计算机辅助审计中的应用研究[J]. 计算机应用研究，2008，25(6)：1908-1910.
[8] 刘汝焯，等. 审计数据的多维分析技术[M]. 北京：清华大学出版社，2006.
[9] 刘汝焯，等. 审计分析模型算法[M]. 北京：清华大学出版社，2006.
[10] 刘汝焯，等. 计算机审计质量控制模型[M]. 北京：清华大学出版社，2005.
[11] 刘汝焯，等. 计算机审计技术和方法[M]. 北京：清华大学出版社，2004.
[12] 董化礼，刘汝焯，等. 计算机审计案例选[M]. 北京：清华大学出版社，2013.
[13] 董化礼，刘汝焯，等. 计算机审计数据采集与分析技术[M]. 北京：清华大学出版社，2002.
[14] 国家 863 计划审计署课题组. 计算机审计数据采集与处理技术研究报告[M]. 北京：清华大学出版社，2006.
[15] 何玉洁. 数据库基础及应用技术[M]. 2 版. 北京：清华大学出版社，2004.
[16] 风笑天. 社会学研究方法[M]. 北京：中国人民大学出版社，2009.
[17] 乔鹏，李湘蓉. 会计信息系统[M]. 北京：清华大学出版社，2002.
[18] 姚泽泓，王海文. 河北省审计厅预算执行审计运用大数据分析纪实[EB/OL]. [2018-02-12]. http：//www.cbdio.com/BigData/2017-02/14/content_5449016.htm. 2017-02-14.
[19] 余卓文. 依托大数据审计技术实现"大海捞针"[EB/OL]. [2018-02-02]. http：//www.gxaudit.gov.cn/show.php? contentid=5703&catid=34. 2017-05-26.
[20] Robert I Kabacoff，高涛，肖楠. R 语言实战[M]. 陈钢，译. 北京：人民邮电出版社，2013.
[21] Rederick J，Gravetter，Larry B，等. Statistics for the Behavioral Sciences[M]. 7 版. 王爱民，李悦，译. 北京：中国轻工业出版社，2008.
[22] Jiawei Han，Micheline Kamber. 数据挖掘-概念与技术(影印版)[M]. 北京：高等教育出版社，2001.
[23] Scott W Ambler，Pramod J Sadalage. 数据库重构(英文版)[M]. 北京：人民邮电出版社，2007.
[24] Tom Soukup，Ian Davidson. 可视化数据挖掘-数据可视化与数据挖掘的技术与工具[M]. 朱建秋，蔡伟杰，译. 北京：电子工业出版社，2004.
[25] W H Inmon. Building the Data Warehouse[M]. 北京：机械工业出版社，2000.
[26] Weber R A. Information System Control and Audit[B]. Prentice Hall. 1999.
[27] Pang-Ning Tan，Michael Steinbach，Vipin Kumar. Introduction to Data Mining[M]. 北京：机械工业出版社，2010.
[28] Jamie MacLennan，Zhaohui Tang，Bogdan Crivat. Data Mining with SQL Server 2008[M].

Indianapolis：Wiley Publishing，Inc.，2009.

[29] Paul Nielsen，Mike White，Uttam Parui. Microsoft SQL Server 2008 Bible[M]. Indianapolis：Wiley Publishing，Inc.，2009.

[30] H V Jagadish，Johannes Gehrke，Alexandros Labrinidis，etc. 大数据及其技术挑战[J]. ACM 通讯，2014，57(7)：86-94.